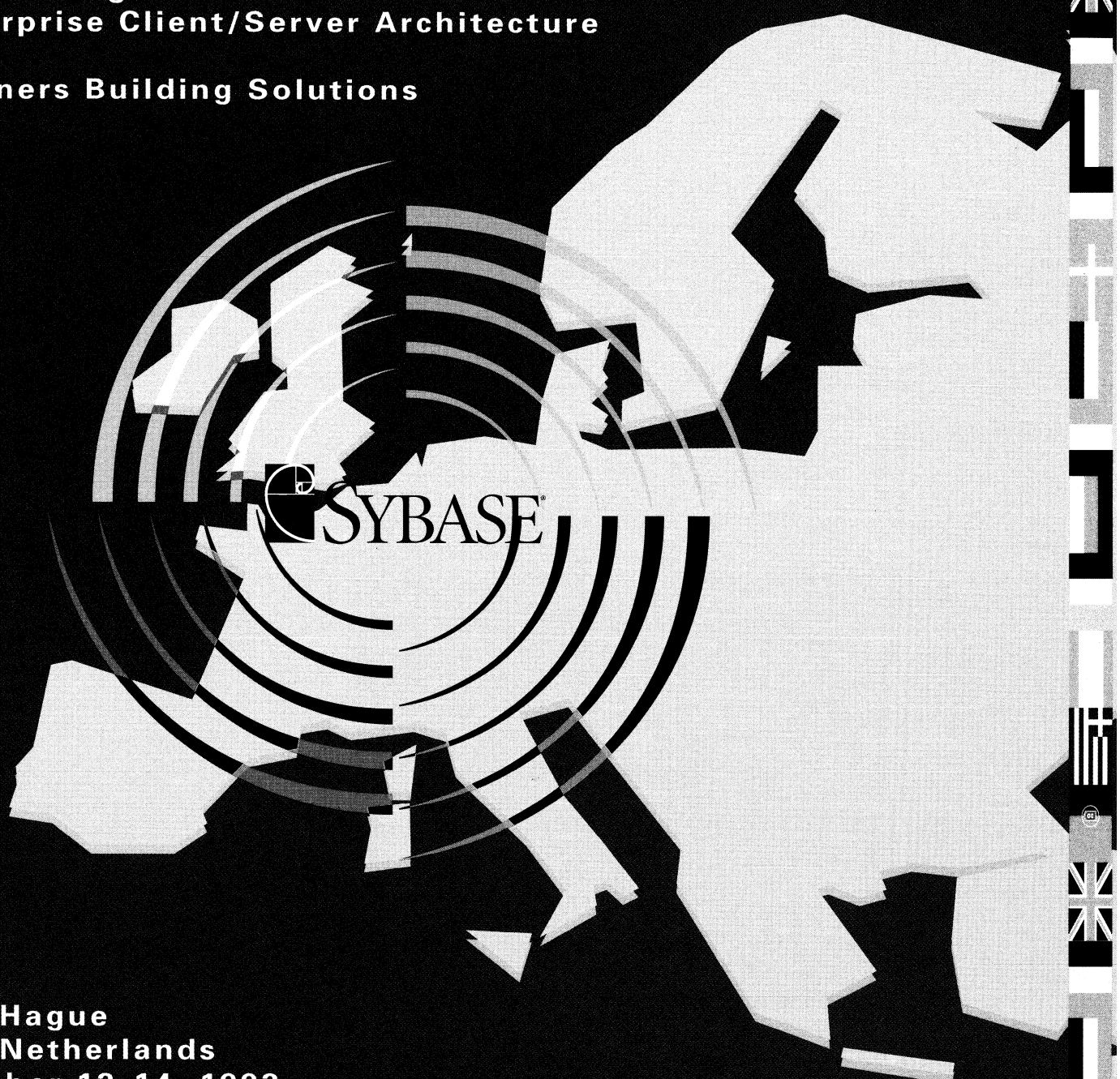
1st annual

SYBASE

european user conference

Introducing the Next Generation of Enterprise Client/Server Architecture

Partners Building Solutions



The Hague
The Netherlands
October 12-14, 1993

TABLE OF CONTENTS
by Track

FOOTNOTES:

* These presentations have not been printed in the proceedings. They will be handed out prior to the presentation.

TABLE OF CONTENTS
by Technical Level

FOOTNOTES:

* These presentations have not been printed in the proceedings. They will be handed out prior to the presentation.

TABLE OF CONTENTS
by Author

FOOTNOTES:

* These presentations have not been printed in the proceedings. They will be handed out prior to the presentation.

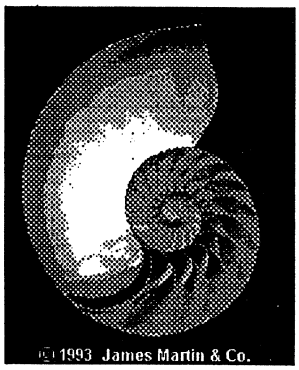
TABLE OF CONTENTS
by Title

FOOTNOTES:

* These presentations have not been printed in the proceedings. They will be handed out prior to the presentation.

Keynote Address

Presentation



Sybase European User Conference - The Hague 1993

Enterprise Client Server:
The boundary between
BRE and rightzing!

Gerard A.M. Otten RI FBCS
James Martin & Co.

Agenda of this presentation

- Developments in the Business Environment
- Objectives for Business Re-engineering
- Requirements for "Right-Sized" Information Technology Responses
- Client/Server Pitfalls and Risks
- Roadmap to Enterprise Client/Server Computing
- The SYBASE Products - Conclusions

Developments in the Business Environment - The Agenda

- The Challenge
- Integrated Change Processes
- Enterprise Engineering Assessment
- Strategic Visioning

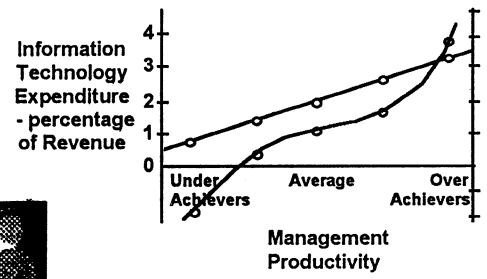
Common Challenges

BARRIERS	KEY "HOW TOs"
Lack of Shared Vision	Shared Vision Process
Persistence of Old Culture	Partnership Mentality
Complacency	Aggressive Management Goals
Inappropriate Performance Metrics	Link Goals and Measures
Failure to Learn from Experience	Focus on Lessons Learned
Lack of Global Flexibility	Avoid Parochial Thinking
Lack of Cycle Understanding	Teams and Market Orientation

IT Barriers

- Fragmented data
- Current systems
- Development culture
- Management inertia

I.T. Revenue and Management Productivity



Technology Trends

- Pervasive, powerful intelligent devices
  - Computer ——> Multiple devices
- Client Server and Right Sizing
- Cooperative Processing ("Groupware")
- Changing Network Paradigm
  - Always Connected
  - Shared Intelligence (User and Vendor)
  - Openness and Differentiation

Technology Trends

- Image, Voice, enabled by Object Orientation
- Electronic Channels
  - Biased ——> Un-biased ——> Personal
  - Sharing with customers, No intermediaries
- Rapid Development, Time to Market

Structural Trends

- Hierarchical ----> Networked Organisation
- Blurred Industry Boundaries:
  Planetary" ----> "Plasma"

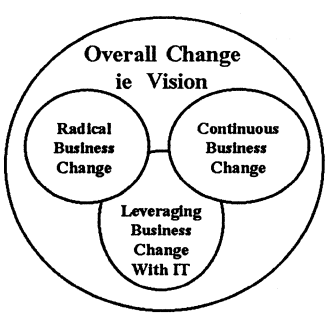
"We are a highly distributed client server organisation. We need the right technology to support the business."

Pat Lupo, CEO of DHL

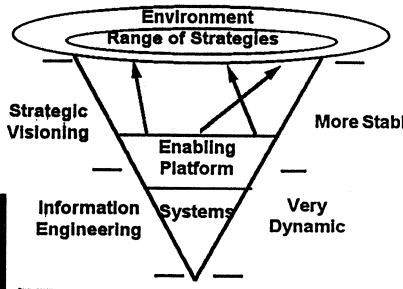
The New Visions

- Quality - "Customer Delight"
- Responsiveness
- "Strategic opportunities from IT"
- "Empowering the workforce"
  - technology support
  - client server
  - right sizing
- "Change" "Survival"

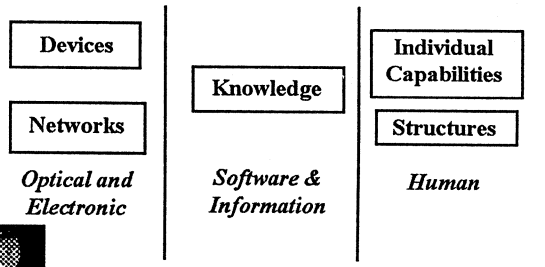
Enterprise Change Goals



Vision, Environment and Strategy



Information Infrastructure



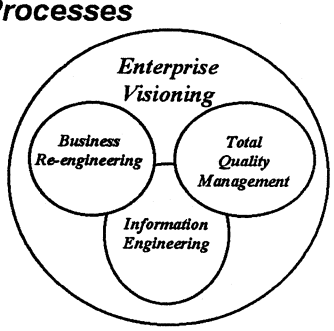
Information Infrastructure

- Orderly
- Responsive
- Seamless
- Extended
- Empowering

Developments in the Business Environment - The Agenda

- The Challenge
- Integrated Change Processes
- Enterprise Engineering Assessment
- Strategic Visioning

Four Integrated Change Processes



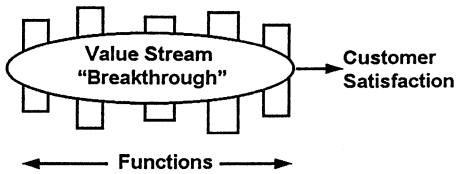
Key Questions Concerning Change

Visioning	Do we need to change? How much needs to change?	Model of Change
Building	How do we change?	Building Processes
Controlling	How are we doing?	Management Processes

Total Quality Management

A systematic approach taken by all employees to achieve the highest levels of quality and competitiveness through continuous improvement of operations, products and services

Business Re-Engineering

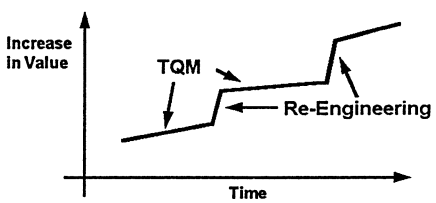


Total Quality Management
Business Re-engineering

Value Stream

"A value stream is a logical collection of people, skills, tools and tasks that interact to promote customer satisfaction and thus provide value to the business"

TQM and BRE - Working Together



Social Systems Impact

- Deals directly with management perceptions of critical enterprise problems
- High potential for defensive reactions
- Organizational anxiety level raised by project of radical change
- Project team needs sponsor reassurance
  - Their efforts will be useful
  - No retribution or negative consequences

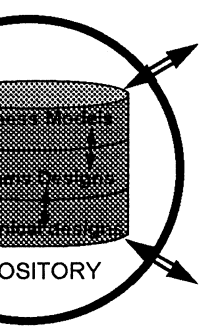
. . . or they cannot freely create new ideas

Role of Repository in EE

- EE - reinvents the business
- IE - systems change to support the new business

Repository:

- Controls and coordinates
- Minimises change and maximises reuse



Agenda of this presentation

- Developments in the Business Environment
- Objectives for Business Re-engineering
- Requirements for "Right-Sized" Information Technology Responses
- Client/Server Pitfalls and Risks
- Roadmap to Enterprise Client/Server Computing
- The SYBASE Products - Conclusions

Business Re-engineering

The fundamental analysis and radical redesign of the business processes, jobs, structure and control to achieve dramatic performance improvement in cost, quality, service and speed.

A Comparison

Key Element	Information Engineering	Total Quality Management	Business Re-Engineering
Obsession	High Quality, fast change, low cost	Quality, attitude to customers	Breakthrough
Customer	Involvement	Satisfaction	"Satisfier" driven
Process	Analyse to share information	Analyse to measure	Analyse to improve
Improvement	Prioritised	Constant, reduce reduce variances	Clean slate
Organisation	Unity of purpose, top down, continual education	Unity of purpose, top down, continual education	Projects, top down, team education

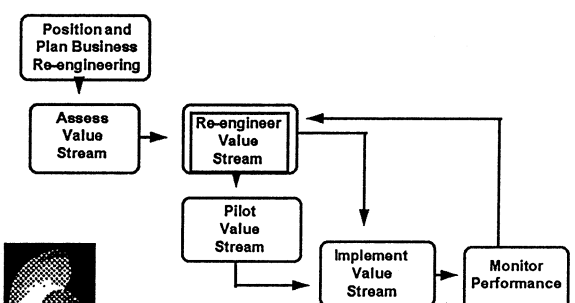
The BRE Perspective

- Set very aggressive targets
- Take the customer's view (value stream)
- Give customers excellence
- Empower workers
- User technology innovatively
- Share data across functions
- Develop new measures and rewards
- Rewrite the business rules

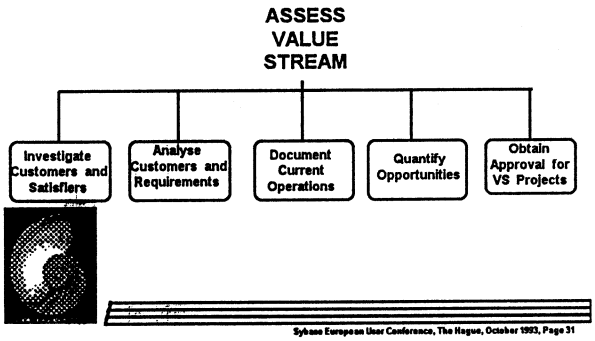
Business Re-engineering - Major Benefits

- Increased Revenue and Profit
- Decreased Cost of Operations
- Increased Customer Satisfaction
- Improved Performance and Scheduling of Business Activities
- Leveraged Competitive Advantage
- Improved Cross Functional Communications

Business Re-engineering Tasks



Value Stream Assessment



What Techniques Are Required for BRE?

- Activity flow and value stream analysis
- Root causal analysis
- Brainstorming
- "Fishbone" analysis
- Structured questioning and interviewing
- Information Engineering modelling techniques
- Benchmarking
- JAD

Benchmarking

- Identify candidate benchmarking companies
- Select industry and non-industry benchmarks
- Develop benchmarking questionnaire
- Conduct benchmarking interviews
- Review results
  - . . . Took one week at Con Edison

Con Edison - Strategy for Business Re-engineering

- Focused on business-driven re-engineering not just business-driven systems
- Introduced facilitation and JAD techniques
- Committed to develop the organisation's business re-engineering skills
- Used Information Engineering to capture re-engineered information needs

Con Edison - Strategy for Business Re-engineering

- Involved key systems and business managers
- Selected two areas for re-engineering
- Set the direction for systems involvement and support

Con Edison Project 1: Procurement

- Mission
  - Review corporate procurement process including organisational involvement, procedures and technology
- Objective
  - Determine required changes to assure all personnel can obtain needed materials and services efficiently
- Phase 1
  - Assess and Re-engineer (3 months)
- Phase 2
  - Detailed Design and Pilot Implementation (6 months)

Procurement Project: Re-engineering Expectations

- Eliminate work that adds no value
- Speed up the procurement process
- Reduce frustrations
- Improve planning
- Promote standardization
- Manage resources effectively

Procurement Project : Results

- Eliminate purchase authorisation process
- Reduce purchase orders and invoices by 80%
- Provide on-line catalogue ordering
- Report order and receipt electronically
- Pay on receipt
- Establish long-term contacts
- Commit vendor to maintain inventory

Con Edison: Sacred Cows

- Authorisations
- Invoices
- Local Orders
- Restricted Items
- Control by Bottleneck
- Requisitions
- Test / Inspections
- JIT Inventory
- Vendor as Bandit
- Short Term / Low Bid

Con Edison: Annual Cost Avoidances Due to Procurement Re-engineering

- Cost of processing PO related invoices
  179,179 @ $15 = $2,687,685
- Cost of preparing N.S.POs under $5,00
  14,493 @ $50 = $724,650
- Cost of responding to referrals
  58,388 @ $25 = $1,459,700
- Cost of creating requisitions
  25,071 @ $25 = $626,775
- Total > $5.3 million

Con Edison Project 2 : Financial Management

Mission:

- Determine what changes are required to assure that all organisations are able to efficiently obtain and process the financial information required to meet their business objectives

. . . BRE is an ongoing process

. . . Continuous Improvement

What are the Critical Success Factors for BRE?

- Led from the Top - Sponsorship, Involvement
- Desire for Radical Improvement
- Motivate Project Team (Full Time)
- "Clean Sheet of Paper" - no preconceived ideas
- Value Stream, not organisational, orientation
- Clear distinction between incremental process improvement and re-engineering
  - Re-invent, without constraints
  - Give us options to scale down

Agenda of this presentation

- Developments in the Business Environment
- Objectives for Business Re-engineering
- Requirements for "Right-Sized" Information Technology Responses
- Client/Server Pitfalls and Risks
- Roadmap to Enterprise Client/Server Computing
- The SYBASE Products - Conclusions

Work Patterns are Changing

- Movement towards self-management
- Integration of work groups with support departments
- Supervisory role disappears or becomes one of facilitation
- The missing "X-factor" - Trust

Tom Peters - *Liberation Management*

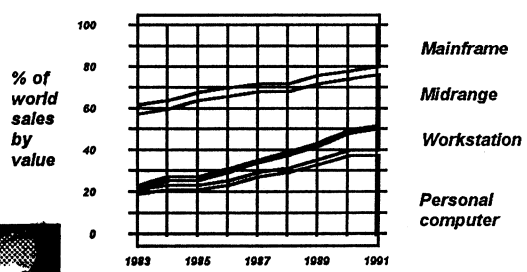
The Cluster Organisation

- A group of people drawn from different disciplines who work together on a semi-permanent basis
- Clusters:
  - » develop expertise
  - » express strong customer orientation
  - » push decision making toward the point of action
  - » share information broadly
  - » handle administration
  - » accept accountability
- PC networks - the infrastructure which underlies clusters

Corporate IT - a Changing World

- New Needs for IT Support
  - ❖ On the move
  - ❖ Multi time-zone
  - ❖ Networking
  - ❖ Speed of decision making
  - ❖ Availability of digestible information
- Appropriate Technology
  - ❖ PC's
  - ❖ Workstations
  - ❖ Networks and Servers
  - ❖ Mainframes

Hardware Sales Trends



PROCESSOR REVENUES BY TYPE

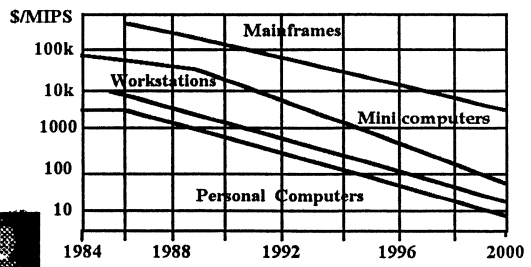
Hardware Sales Trends

- Mainframe sales % has halved
- Mid-range sales % is down too
- PC sales % has levelled off (soon levelling off in volumes too?)

But . . .

- Workstation sales are booming!

Rightsizing Cost Trends



Paradigm Shift

A fundamental change in technology or techniques that alters our perception of systems

Examples:
- The change from telegraph to wireless
- The change from batch to TP
- The change to Information Engineering and I-CASE

Most people do not comprehend or believe in a paradigm shift until after it has happened

A Decade Of Paradigm Shifts

"The 1990s is a decade of paradigm shifts in Information Technology. An enterprise failing to master these shifts fails to upgrade its basic machinery for competition."

Dr. James Martin

Paradigms Shifts of Strategic Importance

FROM	TO
Fragmented Networks	Corporate-Wide Integration
Corporate Systems	Intercorporate EDI
Back-Office Automation	Mission-Critical Systems
Analysing Current Systems	Inventing Strategic Systems
Automating Current Procedures	Business Process Redesign
No User Involvement	User-Responsible Development
Systems Engineering	Enterprise Engineering
Mainframe	Client Server

The Modern-style Enterprise

YOU CAN'T BUILD IT SAFELY WITHOUT METHODOLOGY SUPPORTED BY CASE

- User Oriented
  - GUI
  - Image processing
  - Client-server
  - Downsized
- Architecturally based
  - Sharing data across applications
  - Applying EDI
  - Fully distributed
  - Integrated systems for the integrated enterprise

Make the enterprise more Competitive

- Better customer service
- Higher quality products
- Shorter delivery cycle
- More responsive to customer
- JIT techniques
- Streamlining; elimination of some procedures, departments
- Cost savings: less paperwork, staff, telephone time

Make the enterprise more Capable

- Growing knowledgeworker skills
- Building corporate know-how
- Shorter design cycle
- Better product design
- Better information to decision-makers
- Better planning, market research
- Links to customers
- Integration across the value chain

Make the enterprise more Adaptable

- Faster reaction time
- Faster manufacturing cycle
- More flexible manufacturing
- Faster response to competitive situations
- Immediate information to top management
- Flatter management hierarchy
- Ability to change products quickly
- More customised products

Corporate IT Strategy needs Three Strengths

- Business Acumen
  - Understand competitive advantages via IT
  - Strategic system vision
  - Establish business plans for IT
- Management strength
  - Establish conformity among divisions
  - Enforce standards and building codes
  - Negotiate with users
- Technical knowledge
  - Understand future trends
  - Select architectures
  - Set standards and building codes

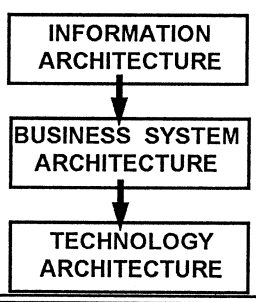
Enterprise Characteristics which affect the choice of Architecture

- The Enterprise
  - Demographics
  - Degree of autonomy of divisions
  - Merger and takeovers
  - Patterns of change
- Its Technology
  - Diversity of computer centres
  - Diversity of existing networks
  - Form of distributed computing

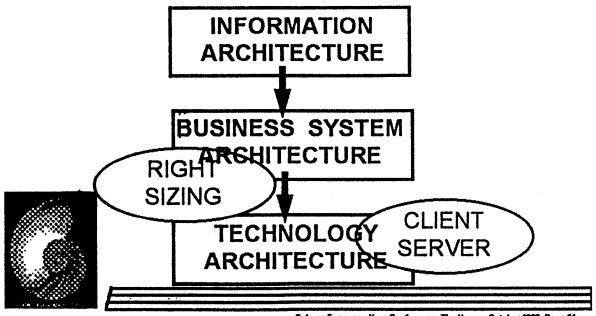
Planning the I.T. Platform

- Examine enterprise characteristics
- Determine business opportunities
- Assess how platform technology will change
- Set objectives
- Determine strategies
- Make architecture decisions
- Determine how to enforce architecture decisions
- Determine short-term tactics

Deriving the Architectures



Deriving the Architectures



Right Sizing and Downsizing

1970s: Mainframe + terminals	One size!
1980s: PC's wherever possible	Downsizing
1990s: The best mix	Right Sizing

Right Sizing is not just Downsizing

- But downsizing is a key element of right sizing
- Get things off the mainframe if they do not belong there
- How do you know?

Right Sizing is not just Downsizing

- But downsizing is a key element of right sizing
- Get things off the mainframe if they do not belong there
- How do you know?
- Plan, analyse, decide, design

Downsizing

INCREASE THE EFFECTIVENESS OF BUSINESS SYSTEMS BY:

- Moving certain data and processes from mainframes to the desktop
- Moving certain data and processes from mainframes to LAN - servers
- Employing compatible, low-cost, best-of-breed engines

Downsizing: Why Do It?

- Decrease hardware costs
- Decrease software costs
- Decrease development costs
- Increase flexibility
- Support rapid change
- Bring computing to its users

Client Server is a key element of Down Sizing

- Everything is in its place
- Everything is available to everyone who needs it

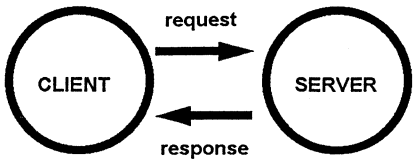
But . . .

- How do you know who needs what?
- How do you share data?
- How do you resolve conflicting needs?

INFORMATION ENGINEERING

What is Client Server?

- Client initiates request and receives service
- Server receives and executes request
- Two sets of code



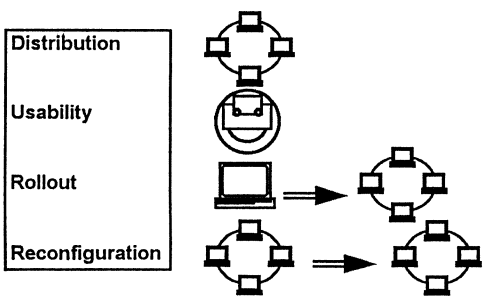
Client Server Definition

- Clear functional separation between client and server
  - . . . *but seamless interaction*
- Client and server can operate on separate machines
  - . . . *but don't have to*
- Server can support multiple clients concurrently
- Server can be changed without impacting clients
  - . . . *for scalability, availability*
- Client can be changed without impacting other clients
  - . . . *they are independent and tailorable*

Client Server Definition

- Clear functional separation between client and server
  - . . . *but seamless interaction*
- Client and server can operate on separate machines
  - . . . *but don't have to*
- Server can support multiple clients concurrently
- Server can be changed without impacting clients
  - . . . *for scalability, availability*
- Client can be changed without impacting other clients
  - . . . *they are independent and tailorable*

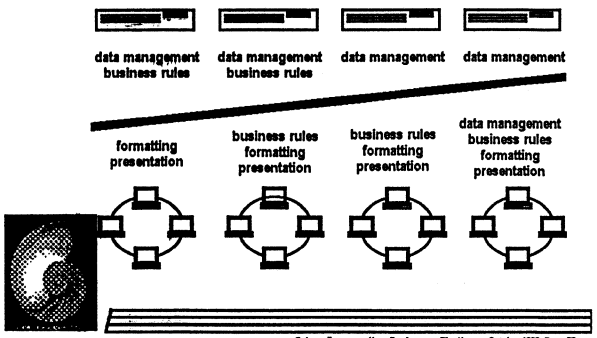
Key Elements of Client Server Design



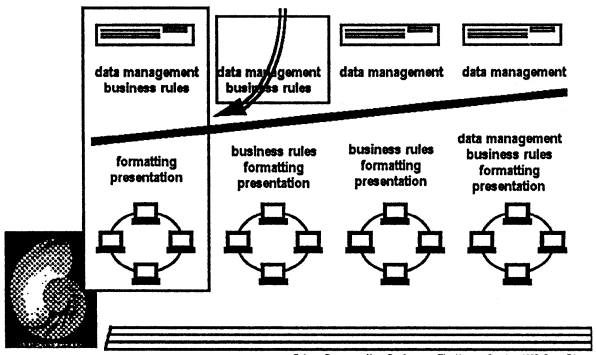
Distribution

- Data
  - Where is the data held?
  - Replication
  - Vertical or horizontal fragmentation
- Process
  - Where should the process run?
  - Where are the results needed?
- Transaction management
  - Concurrency
  - Bundling transactions to optimise use of protocol
  - Performance management

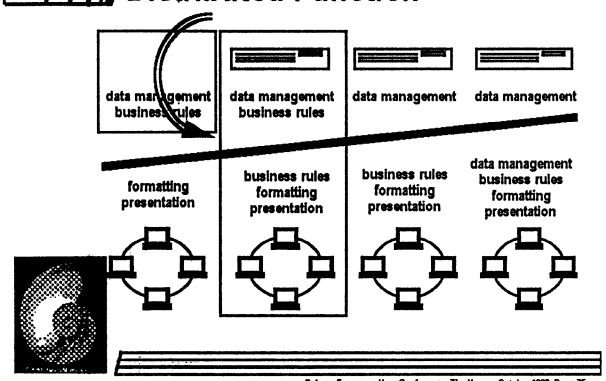
Level of Distribution



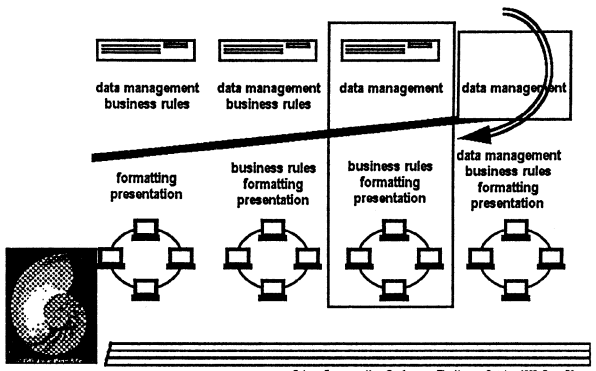
Remote Presentation



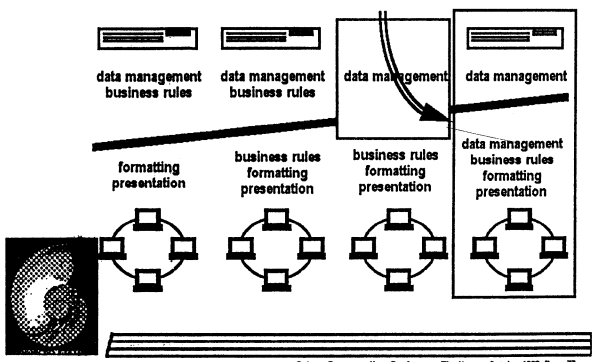
Distributed Function



Remote Data Management



Distributed Database



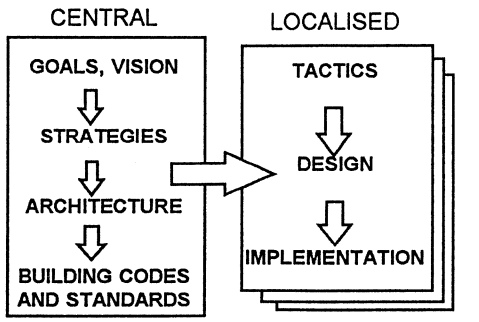
Architecture Selection

- Decide appropriate architecture at corporate or application level
- Consists of
  - Client hardware and OS
  - Server hardware and OS
  - Network implementation and OS
  - DBMS (server, client)
  - Transaction management software
  - GUI support
  - Configuration management

Agenda of this presentation

- Developments in the Business Environment
- Objectives for Business Re-engineering
- Requirements for "Right-Sized" Information Technology Responses
- Client/Server Pitfalls and Risks
- Roadmap to Enterprise Client/Server Computing
- The SYBASE Products - Conclusions

Positioning IT Responsibilities



Organisational Implications

CENTRALISE:

- Corporate strategy
- I.T. strategy
- Enterprise re-engineering
- Architectures for Information Engineering
- Redesign of the value streams

Organisational Implications

CENTRALISE:

- Architecture for
- Business modelling
- Application integration / Data sharing
- Corporate network
- Connectivity (SAA, NAS, etc)
- Distributed computing (DCE, DME)
- Database access (Info Warehouse)

Organisational Implications

CENTRALISE:

- Coordination for
- CASE technology, Repository, Reusability
- Building codes and training
- Project scoping and interaction
- Education and cultural change

Organisational Implications

DECENTRALISE:

- Departmental process redesign
- Localised building of systems
  - local understanding and ingenuity
- RAD projects
- Downsized systems (LANs, etc.)

Desirable prerequisites for Rightsizing

- Common network architecture
- Strategy for portability
- Relational Databases; Common SQL
- SAA, UNIX, Windows NT or equivalent
- CUA or other GUI standards
- Client-server management software
- IE methodology
- Appropriate CASE tools
- Development coordination function

Inhibitors to Rightsizing

- Mainframe culture
  - Many I.S. professionals understand the mainframe environment but not the PC-Server environment
- Cost of rebuilding applications
- Lack of CSTP management features in software
- Lack of I-CASE tools
- Additional complexity

Beware Costs

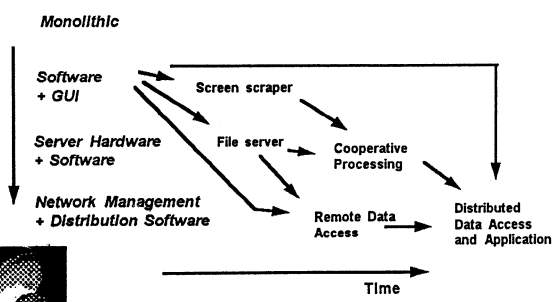
- Distributed environment
- Testing is complex - GUI and distribution
- Immature and more complex software
- Even greater demand that applications be close to zero defects - business fit, software quality

Right Sizing - Summary

- The principle of "centralise / decentralise" puts appropriate technology on desks and in workplaces to achieve:
  - User driven systems
  - Optimum effectiveness
  - Maximum efficiency

  for the business
- For success it must be approached strategically - which means as part of the business strategy

Migration to Client Server



How Can Client Server Applications be Developed?

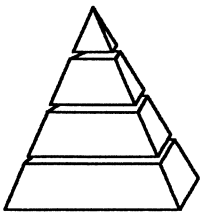
- Client server is complex and powerful
- Risks attached to uncontrolled development
- Cut down risk and focus on quality
- Understand the distribution requirements
- Structure the project by use of methodology

What is a C/S Methodology?

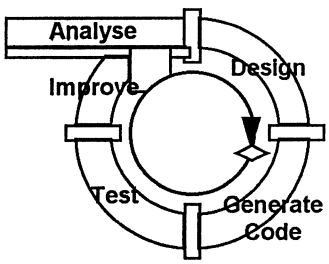
- Methodology:
  - A set of tasks supported by disciplined techniques that, when properly performed, produce a desired result
- I-CASE methodology:
  - A methodology with integrated techniques which use tools and a common repository to provide maximum automation of the development process
- Repository:
  - A precisely structured knowledgebase containing all the facts about the business, its systems and its technology, as gathered during system development and maintenance.

Principles of IEM

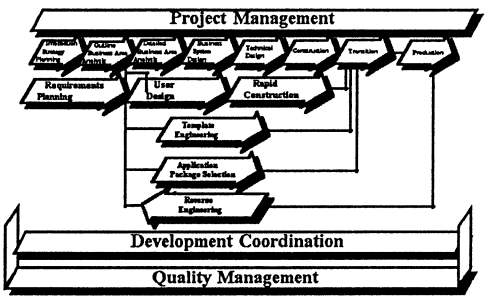
- Business Driven
- Top Down
- Integration/Coordination
- Architectures
  - information
  - business system
  - technical
- Quality
- Multiple Development Paths
- User Involvement



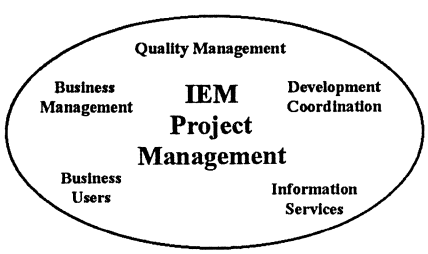
Iterative Development Method



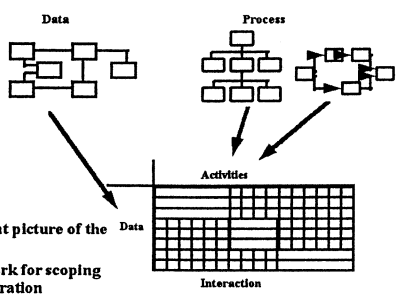
Principles of IEM



IEM Quality and Project Management



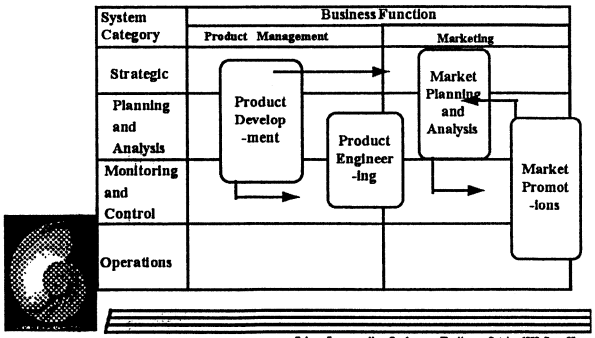
Information Architecture



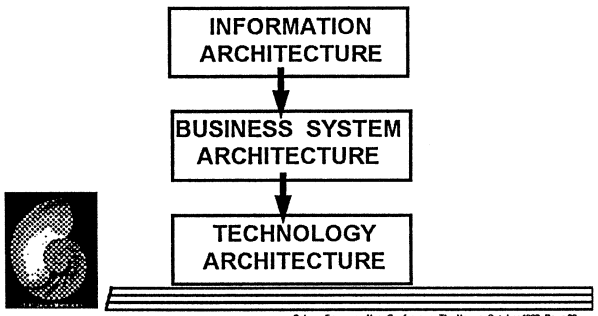
Consistent picture of the business

Framework for scoping and integration

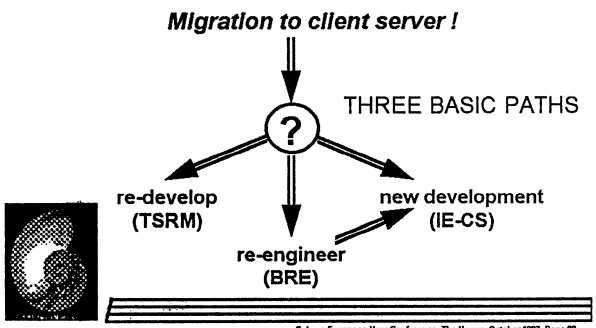
Business System Architecture



Deriving the Architectures

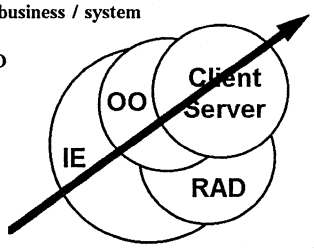


Requirements for a Client Server Methodology



Information Engineering and Client Server: Building on Success

- Information Engineering is proven
- Architectures, standards, business / system separation
- Extended by RAD and OO
- CS is another target architecture which IE can support



Tasks in Engineering of Client Server and GUI Applications

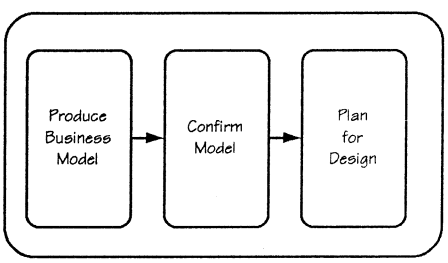
- Colour coding:
  - NEW .....
  - EXTENDED .....
  - STANDARD IE .....

Define GUI Usability Goals

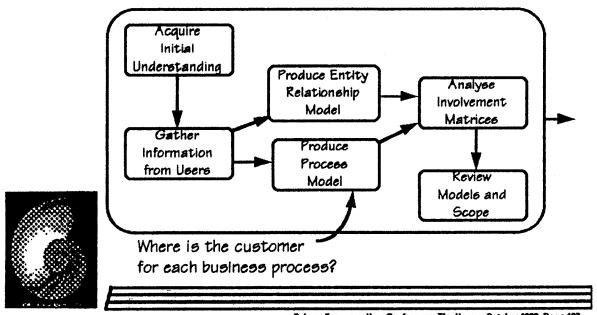
Design Screen / Report Layouts

Verify Design Consistency

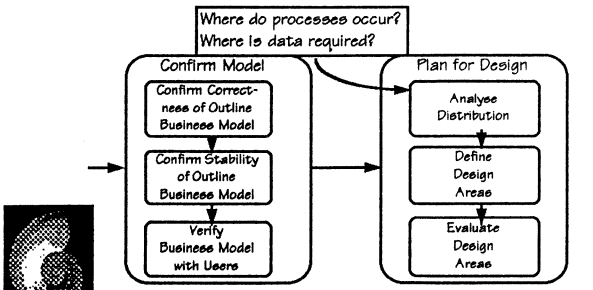
Outline Business Area Analysis -OBAA Tasks



OBAA - Produce Business Model



OBAA - Confirm Model and Plan for Design



Process / Location Matrix

X - Executed
/ - Partly performed
P - Planned performance

	Seoul	Hong Kong	Tokyo	San Diego	Dallas	Supp -lier
Specify Product	X					
Approve Design		X			P	
Negotiate Production	X	P			/	
Evaluate Sample		/	/	/		/

Entity Type / Location Matrix

C - Create
R - Read
U - Update
D - Delete

	Seoul	Hong Kong	Tokyo	San Diego	Dallas	Supp -lier
Product	C	R		R		R
Product Evaluation	U	C	R		U	
Production Agreement	C	R	R			R
Production Sample		U	U	U		C

Business Rules

- Attach business rules to domains
- Choose in design where to enforce it
  - Client or server?
- Use business rules for derivation algorithms
- Use rules for data state-driven triggers
  - Translate into DBMS triggers

Action Diagrams

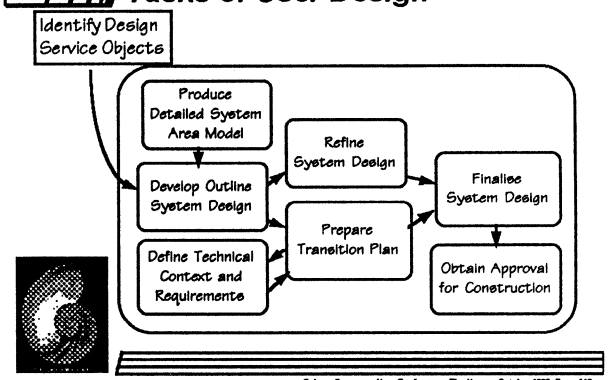
- Take care over placing logic
- Client procedure?
- Server procedure?
- Split?

Action Diagrams

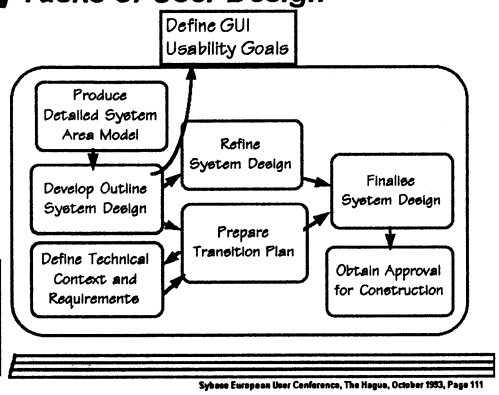
- Take care over placing logic
- Client procedure?
- Server procedure?
- Split?

```
READ Person
  with SSN = Input SSN
  IF Person status = "applied"
    CREATE Interview
      SET date to Input date
      SET comments to Input comments
      ASSOCIATE Interview with person
        which is subject of Interview
  ELSE
    EXIT
Note:Interviews are not performed after
     a person has been evaluated, hired,
     or rejected.
```

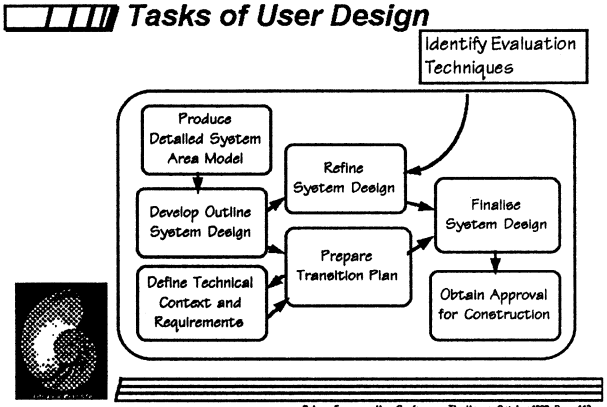
Tasks of User Design



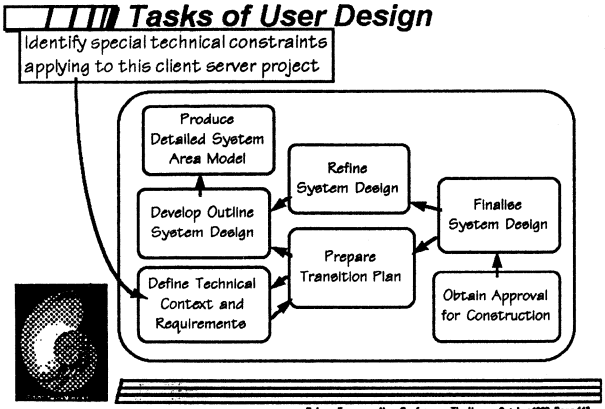
Tasks of User Design



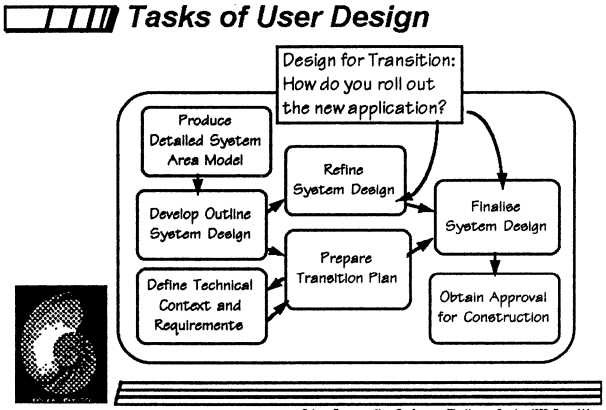
Tasks of User Design



Tasks of User Design



Tasks of User Design



Process to Procedure Mapping

- Elementary process - rigorous definition
- If not elementary:
  - Too large
    - » - opportunities for distribution missed
  - Too small
    - » - integrity risk, commit unit impossible to specify

Typically one-to-many mapping

- Never one-to-one for client server
- Generic division of process into procedure elements:
  - Presentation logic
  - Business logic
  - Database logic

Network and Distribution Design

- Highly iterative
- To minimise traffic and cost
- To maximise efficiency
- Covers the distribution of
  - Process
  - Data
  - Transaction management
- Distributed transaction testing to analyse and optimise performance

Design for Transition

- Orders of magnitude more complex in a client server environment
- Test elements of transition in the design process
- Design not finalised until transition issues addressed
- Plan early for migration, user acceptance, rollout

Usability Failures

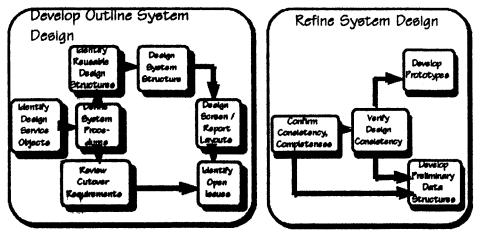
- Poor GUI design is a major reason for client server failure
- The answers include
  - Human engineering
  - Cognitive Factors
  - Usability Testing

User Design - Usability Engineering - Specific Tasks

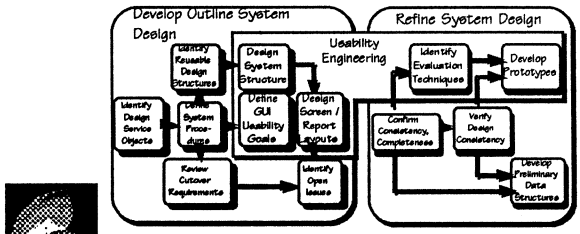
GUI design requires specific Usability Engineering tasks and techniques

- In Develop Outline System Design ☐
  - Design System Structure ☐
  - Define GUI Usability Goals ☐
  - Design Screen Layouts
- In Refine System Design ☐
  - Identify Evaluation Techniques ☐
  - Develop Prototype

Usability Engineering in the Task Structure



Usability Engineering in the Task Structure



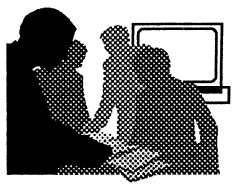
Usability Engineering - Techniques

- Intensive Design Sessions
- Prototyping
- Usability Specification
- Usability Testing

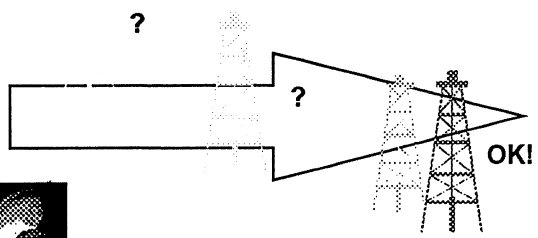
- All of these are iterative tasks ...
  ... *adopt a TIMEBOXING approach*
  ... *get the project management right*

Intensive Design Sessions

- Vital to involve users in prototyping the interface
- To discover:
  - Cumbersome ways to perform basic functions
  - Language problems in menu selection
  - Poor documentation
  - Inconsistent basic functions
  - Poor screen design aesthetics

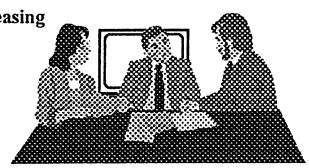


Prototyping



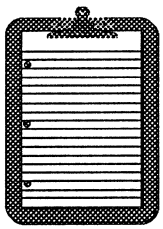
Usability Specification

- Easy to learn
- Efficient to use
- Easy to remember
- Not prone to error
- Subjectively pleasing



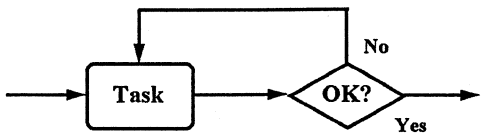
Usability Testing

- Test user reactions to:
- Installation
- Training
- Documentation
- On-line help
- Interface design
- Error recovery



Timeboxing

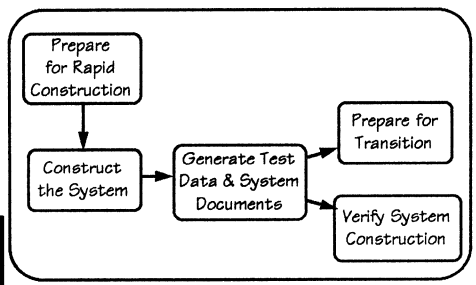
* Fixed time period to achieve a result



. . . Is the quality OK at the end?

. . . May have to re-iterate

Tasks of Rapid Construction



Network Configuration and Testing

- Normal project delivers a DBMS schema
- CS project must also deliver a set of network configurations
- Testing - must aim for close to zero defects
- Complexities of testing:
  - GUI testing
  - Network configuration testing
  - Recovery testing for distributed environment
  - Single and multi-point failures
  - . . . and more!

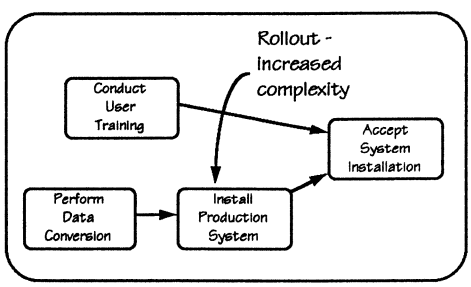
Construct for Re-use

- I-CASE and Repository provide re-use facilities
  - Define
  - Build
  - Implement
- 'Blackboxing' (encapsulation)

And remember . . .

- Process management is vital
- Don't alter code - regenerate

Task of Transition



Rollout - Increased Complexity

- Mainframe environment
  - Update software (1 set)
  - Restructure and reload database (possibly 1 or 2)
  - Install (1)
- Client server environment
  - Update server software (may be more than 1 set)
  - Update client software (may be hundreds)
  - Restructure / reload client and server databases (lots!)
  - Install (every single machine)
- A quick, coordinated programme - say overnight

Tools and Methodology for Client Server

- Topics:
  - What is Client Server?
  - Methodology for Client Server
  - Tools for Client Server
  - Summary

What is a Client Server Tool?

- Development tools which:
- Support analysis / design techniques for CS
- Provide GUI design facilities
- Generate code and database for multiple platforms
- Promote sharing between platforms

There is no single criterion for a 'client server tool'

Key Distinction Between Methodology and Tools

- Methodology provides a framework for action:
  - What to produce?
  - How to produce it?
  - Who produces it?
  - When and why?
- Tools facilitate speed and quality by automating methodology tasks and deliverables
- Tools can support methodology, but can never be a substitute for it

Tools for Client Server

- Screen Scraper
  - GUI front end for existing systems
- Application generators
- Reverse engineering tools

- I-CASE
  - Integrated full life cycle CASE

A Mess?

"You want to be able to develop an application once and run it in many target environments, but that is difficult because the operating environments are in such a mess"

(David Fairbairn - Texas Instruments)

Agenda of this presentation

- Developments in the Business Environment
- Objectives for Business Re-engineering
- Requirements for "Right-Sized" Information Technology Responses
- Client/Server Pitfalls and Risks
- Roadmap to Enterprise Client/Server Computing
- The SYBASE Products - Conclusions

The SYBASE Products - Conclusions

- James Martin & Co. did a study of Sybase:
  - The SYBASE products
  - The Sybase organisation
- Report of the study available

The SYBASE Products - System 10 - The Servers

Low Risk	Well proven advanced technology	SQL Server
Scalable	High capacity with low entry cost	Navigation Server
Interoperable	Corporate multi-vendor transparency	Omni SQL Gateway
Reliable	Reliable data and transaction delivery	Replication Server
Controllable	Distributed control from a single location	System Management Products

The SYBASE Products - The Connectivity Products

- Open Client
- Open Server
- Open Gateway
- SYBASE mainframe integration
  - Net-Gateway
  - Open Server for CICS
  - Open Client for CICS
  - Open Gateway for DB2

The SYBASE Products - The Connectivity Products

- Sybase have been "playing" with DEFT for analysis, design, development of application systems
- Sybase has acquired Gain Technology Inc. to enable multimedia applications capability
- Sybase has made a public commitment to provide development tools that will support the development of open, interconnectable systems based on the enterprise client/server architecture

The Sybase Products - Client/Server Architecture

- Visionary statement with strong product support
- Access to existing data servers and legacy system components through the OmniSQL Gateway
- Full relational database support capability through the SQL Server
- Support of distributed database system implementations through OmniSQL Gateway
- Scalable, possibly distributed, relational processing based on massively parallel query and update transactions based on Navigation Server

The Sybase Products - Client/Server Architecture ...

- Distributed synchronisation of the content of multiple databases through Replication Server.
- Management of the distributed environment using Systems Management Products for a variety of operational issues, like Configurator, Backup Server, SQL Monitor.
- Development Support Systems for the rapid creation of complex business applications.

The SYBASE Products - Conclusions

Major findings:

- SQL Server 10 is a strong baseline product for the enterprise client/server architecture
- The worldwide Sybase customer services and support organisation is being enhanced

The SYBASE Products - Conclusions

Major findings ...

- Sybase is continually improving its market image, and we note that Sybase's substance, structure and style are well received by it's clients
- Sybase is currently the fastest growing DBMS company in the world and projections predict the continuation of this growth, and the rise in the relative importance of the database.

Forrester Research Inc.: The software strategy report Vol3 # 9

James Martin & Co. - Statement

James Martin & Co. endorses the SYBASE Enterprise Client/Server Architecture as an important, visionary approach to information systems achieved with practical and robust products that support business applications right across the enterprise

Management

(Business User Stories)

Presentations

JORGE

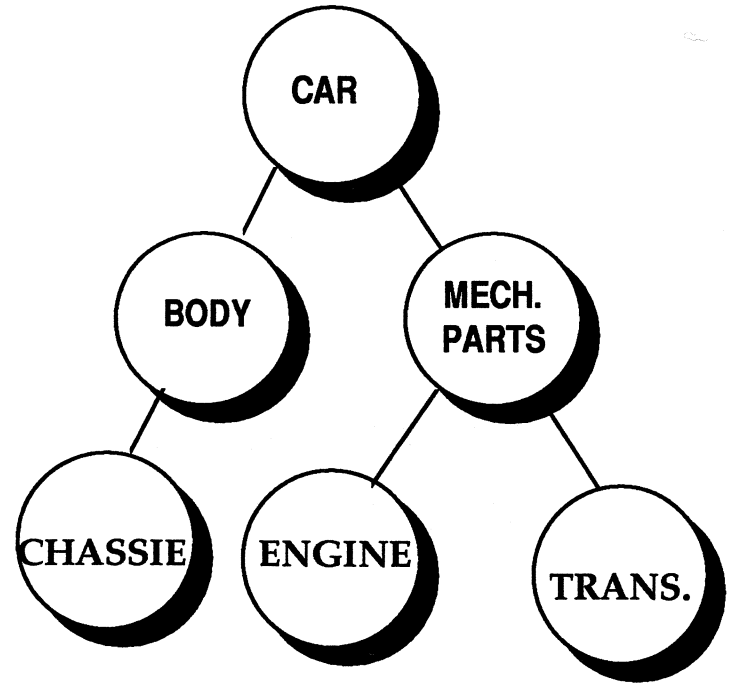
A Client Server Application for Document Management and Configuration Control

Sybase European User Conference, The Hague, 12-14 October 1993

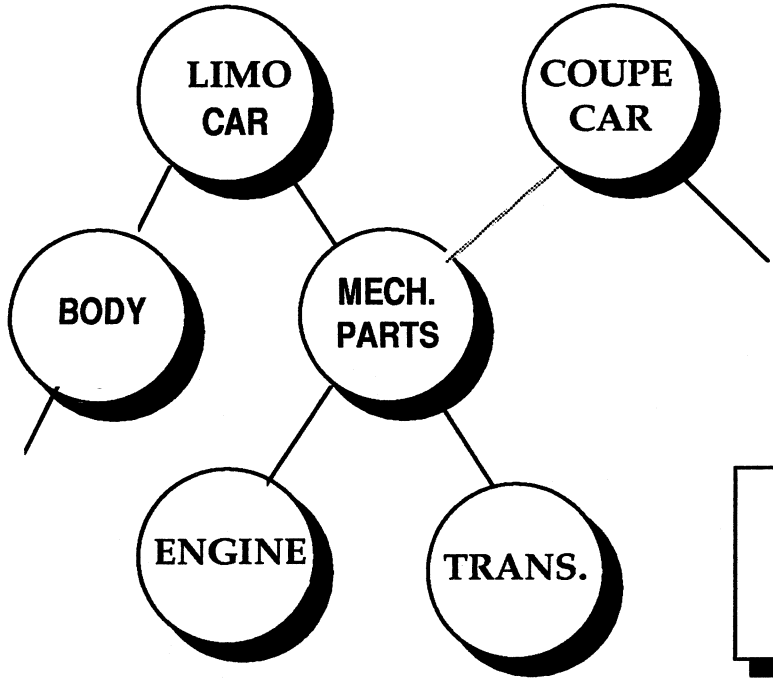
By Gian Paolo Rossi

The Problem

- maintain the relationship existing amongst the parts of a product
- associate the documentation to the configuration items
- ensure consistency and control along the product life cycle



The Problem

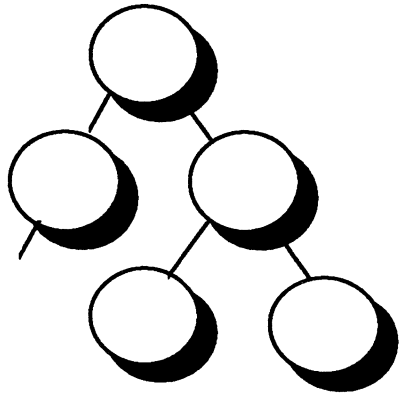


- manage changes, reviews, ...

..... according to enterprise's Quality rules

CONFIGURATION AND DOCUMENT CONTROL

Configuration Management

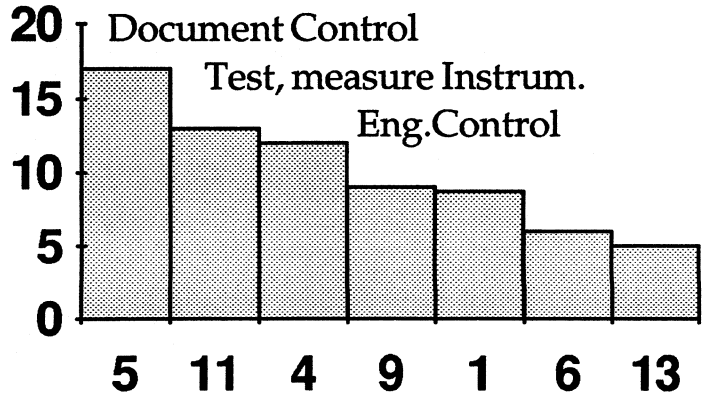


SIDE EFFECTS

- the company takes possession of the control over the information about its products
- efficient interaction amongst people
- reduced time to market
- information is more accessible
- engineering change and document approval cycle get shorter
- maintenance costs are reduced

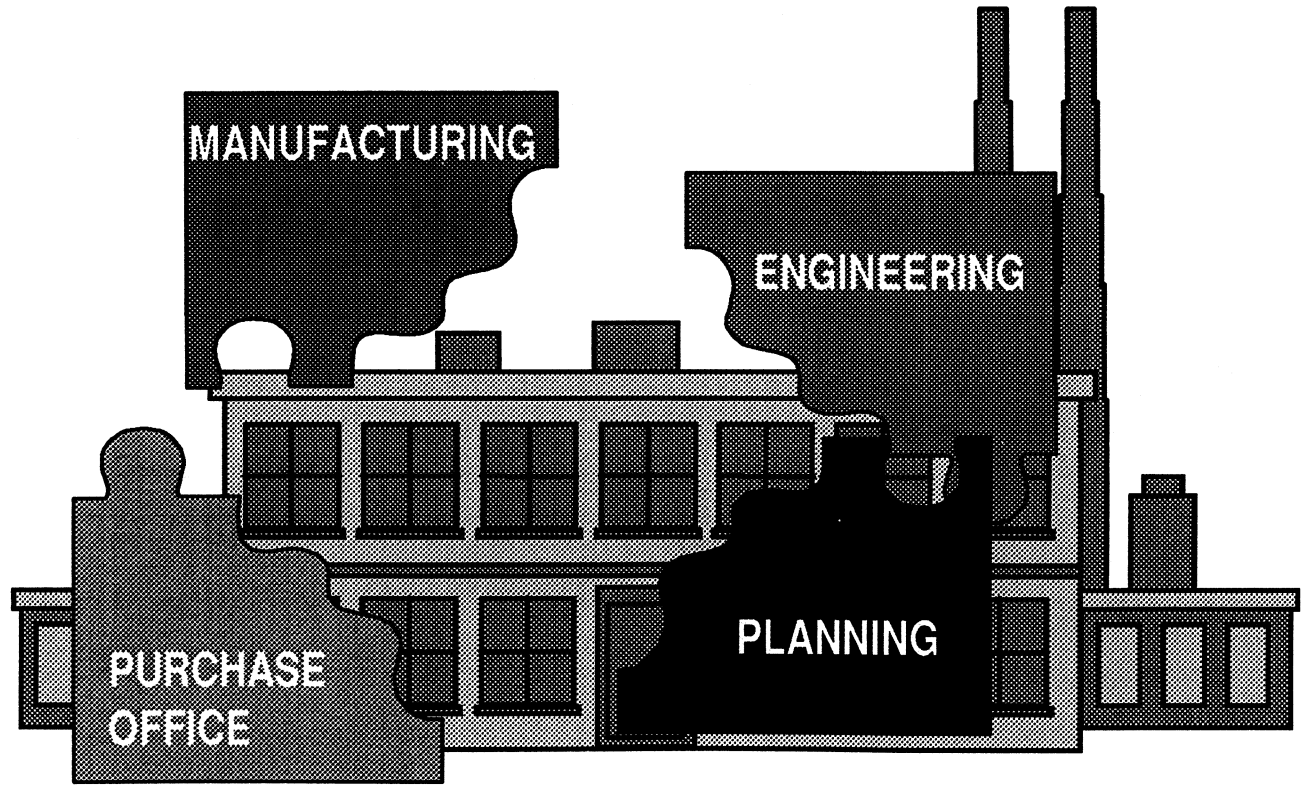
Why a Support to Configuration Management ?

- biggest source of non-compliancies during the certification process (ISO-9000)

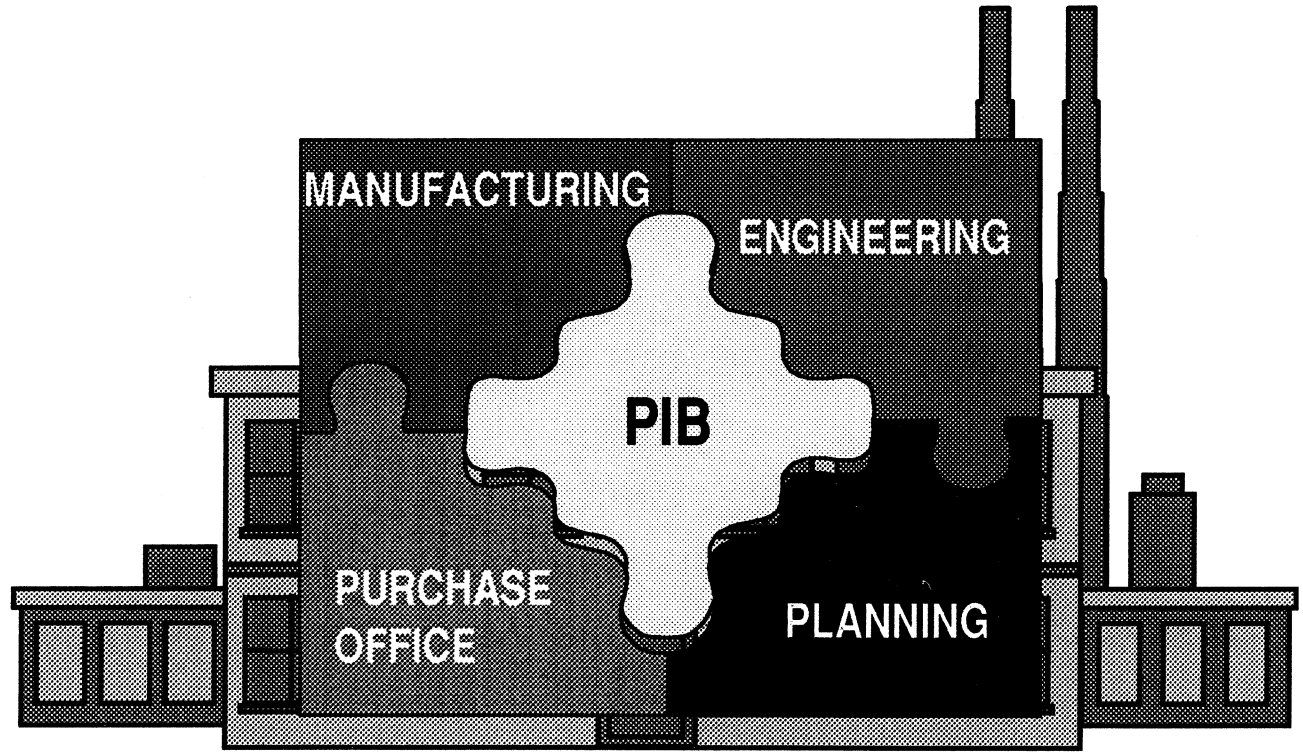


- the Product Information Base is the core element of an integrated company
- very few informatic solutions exist

Enterprise's Integration



Enterprise's Integration



JORGE The Solution

- secure and easy to use solution to the problem
- it captures the enterprise's quality rules for product development, release and change management
- it represents the kernel of an integrated information system
- the management of complex products becomes quicker, more accurate and at lower costs

JORGE Functions

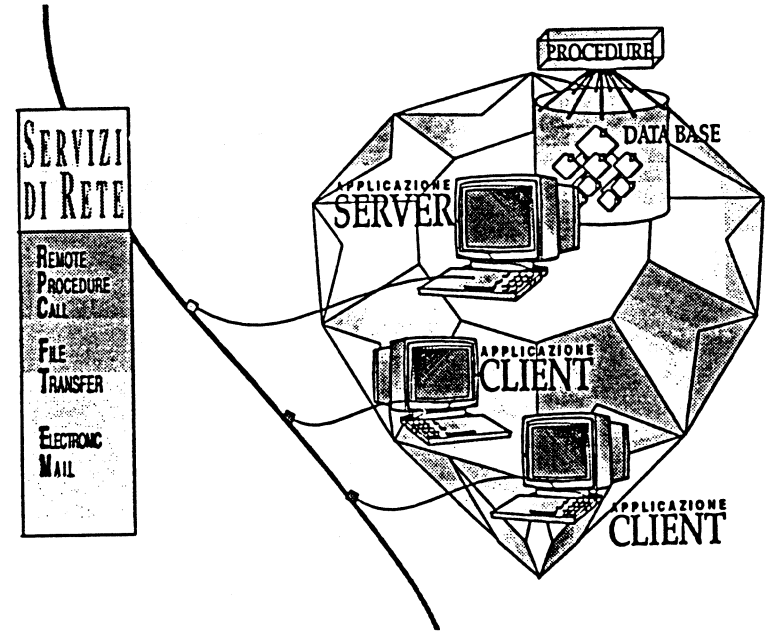
- identify configuration items (c.i.) and aggregate them in the product structure
- product tree control
- import and organize information about c.i.
- document access throughout the network
- document approval and release by means of electronic signature
- engineering change control and procedure

JORGE Functions

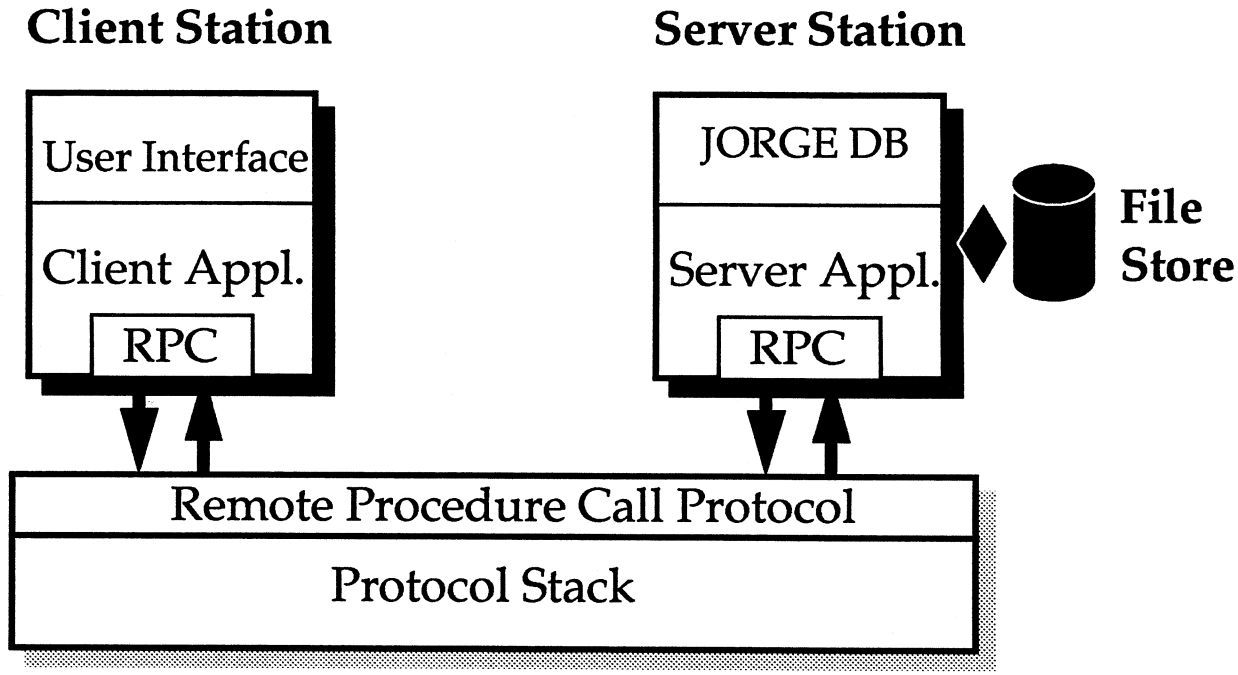
- automatic reporting and integration with other systems
- integrated Part List management
- baseline control
- automatic alert and e-mail management

JORGE Open Architecture

- Client Server Architecture
- Network and Operating System Independency



JORGE The Components



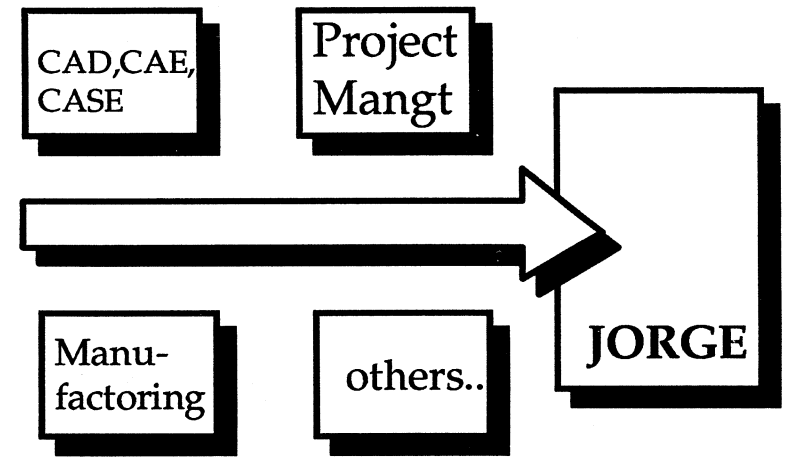
JORGE Technical Notes

Client Station	DOS Windows, UNIX Motif, UNIX OpenLook, Macintosh (**XVT** GUI)
Server Station	UNIX, VMS, OS2
DBMS	**Sybase**
Communication Subsystem	RPC and TCP/IP native complete transparency is achieved by using Sybase Open Solutions

JORGE Independency of Company Size

ALENIA The largest

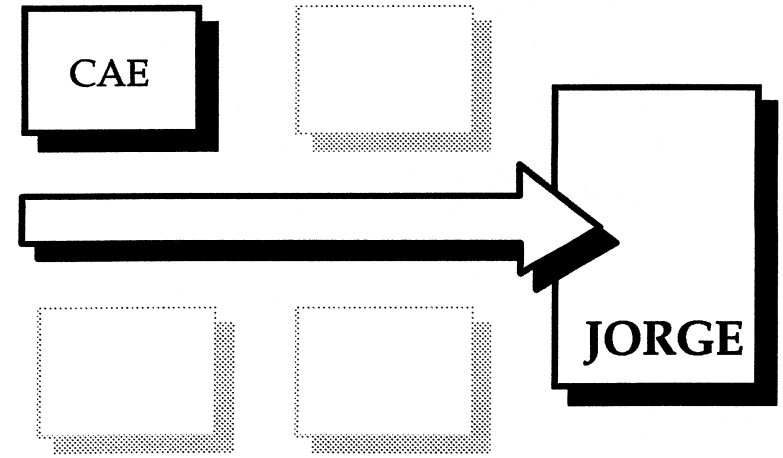
- # 500 users
- # 30.000 documents
- # 5 Gbytes filestore
- # 40 Mbyte data base
- # high level of integration



JORGE Independency of Company Size

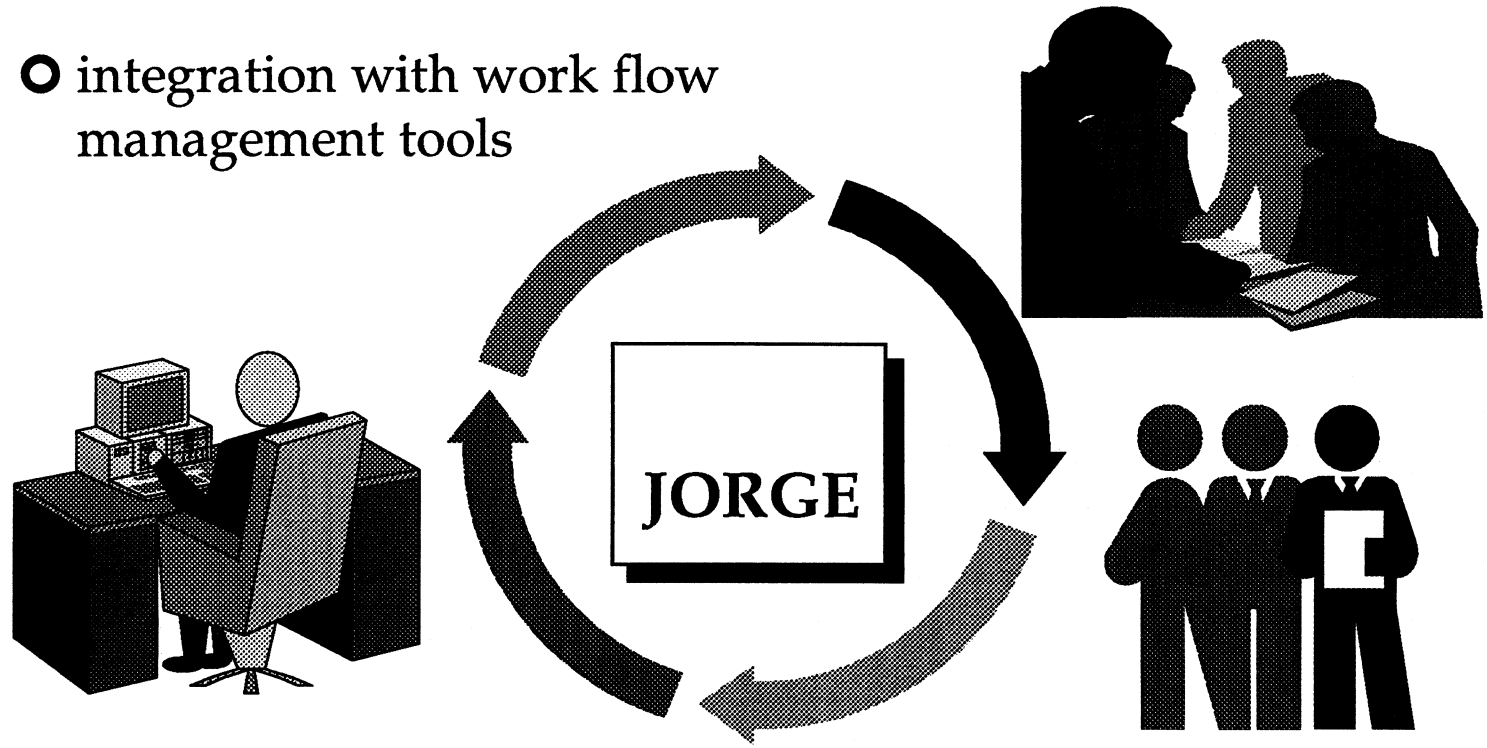
GCAR The smallest

- # 9 users
- # 120 documents
- # 25 products



JORGE Still Growing ...

- integration with work flow management tools



JORGE Concluding Notes

WHERE

- avionic
- electronic
- telecom.
- engineering
- .....

HOW

- low cost
- openness
- consulting

WHO

SYREA and its partners

ACQUISITION STRATEGIES

FOR CLIENT / SERVER

IN THE 90's

JEFF MANDELBAUM
PRESIDENT
SYBASE FINANCIAL SERVICES, INC.

TOPICS

* MYTHS AND FOLKLORE
* WHY SHOULD I INVEST IN CLIENT / SERVER?
* KEY MEASUREMENTS
* MAJOR ACQUISITION STRATEGIES
* SOFTWARE FINANCING
* LIFE CYCLE COSTS (AN EXAMPLE SAVING > US$ 27,000,000)
* SELECTING AND OPTIMIZING VENDOR RELATIONSHIPS
* SUMMARY
* EXAMPLES

MYTHS AND FOLKLORE

MYTH # 1

CLIENT / SERVER BRINGS IMMEDIATE COST REDUCTIONS

MYTH # 2

CLIENT / SERVER PAY BACK IS GENERALLY LESS THAN 2 YEARS

MYTH # 3

CLIENT / SERVER WILL REPLACE MY MAINFRAME IN THE NEAR TERM

WHY SHOULD I INVEST IN CLIENT / SERVER?

1) **TO REDUCE COSTS**

- in IT departments !

2) **TO ENHANCE COMPETITIVE POSITION OR INCREASE MISSION EFFECTIVENESS**

- increasing revenues

3) **THERE ARE NO OTHER BUSINESS REASONS**

KEY MEASUREMENTS

EVALUATE CLIENT / SERVER AS AN INVESTMENT

* **PAY BACK**
* **INTERNAL RATE OF RETURN**
* **NET PRESENT VALUE**
* **PROFITABILITY INDEX**
* **TRADEOFF BETWEEN RISK AND RETURN**
* **STAFF RETENTION AND ENRICHMENT**

↳ get your people motivated

→ Investment to go monthly in technique data client/server

MAJOR ACQUISITION STRATEGIES

TACTICAL

* PROTOTYPES AND PILOT PROJECTS

* (no commitment) VOLUME PURCHASE AGREEMENTS

* RISK FACTORS

STRATEGIC

* MINIMUM PURCHASE COMMITMENTS

* SITE LICENSING

* USER / SEAT BASED PRICING

* RISKS FACTORS

SOFTWARE FINANCING OPTIONS

INSTALLMENT AGREEMENTS / FINANCE LEASES

* CASH FLOW BENEFIT
* AMORTIZE SOFTWARE COST AND EXPENSE INTEREST
* COMPARE VENDOR COSTS WITH YOUR FIXED BORROWING COSTS

OPERATING LEASES OR RENTALS

* CASH FLOW BENEFIT
* NO BALANCE SHEET LIABILITY OR CAPITAL BUDGET REQUIRED
* MAY PROVIDE LOWER AFTER-TAX COST THAN PURCHASE OR FINANCE LEASE
* FLEXIBLE END OF INITIAL TERM OPTIONS

SOFTWARE FINANCING AT SYBASE

* " ... Sybase is the leader in software financing. Their unique structure and creative use of financing added value to our business relationship."

  - Treasurer, LSI Logic Corporation

* " ... Software financing provides great leverage to Sybase revenue and profits. Sybase Financial Services is the envy of the industry."

  - Ed Beck, former President, Meridian Software Funding

PARTIAL SYBASE FINANCING CUSTOMER LIST / INTERNATIONAL DIVISION

La Poste

Prudential Assurance, Ltd.

Accor

Swiss Bank Corporation

Modus

Maestro

Mercury Communications, Ltd.

EDF (Electric Utility of France)

STRONG REVENUE GROWTH

YEAR	SYBASE FINANCED SALES	% Y/Y GROWTH
1990	$10.6MM	N/A
1991	$22.4MM	111%
1992	$57.6MM	158%

LIFE CYCLE COSTS

L not just today

* CRITICAL METRIC TO PLAN BUDGETS AND ASSESS PROPOSALS

* INCLUDE ALL RELEVANT COST FACTORS

* MULTI YEAR COST ESCALATION

* EXPECTED VALUES AND ROI IMPACT

* AN EXAMPLE SAVING MORE THAN US$ 27,000,000 ON A SINGLE PROJECT

SELECTING AND OPTIMIZING VENDOR RELATIONSHIPS

* ANALYZE FINANCIAL STRENGTH AND REPUTATION
  * HISTORICAL FINANCIAL PERFORMANCE
  * ABILITY TO SUSTAIN
  * INTEGRITY
  * INTERVIEW / VISIT EXECUTIVE MANAGEMENT

* LONG TERM PARTNERSHIPS VS. VENDOR / CUSTOMER

* UNDERSTAND / ADDRESS KEY VENDOR ISSUES AND EXPLAIN YOURS

* COMPLETE BUSINESS TERMS BEFORE INVOLVING LAWYERS

* WIN - WIN USUALLY PROVES BEST IN THE LONG RUN

SUMMARY

* CLIENT / SERVER IS AN INVESTMENT

in budget - respective

* SET REALISTIC EXPECTATIONS AND PLAN CAREFULLY

* CLIENT / SERVER REWARDS ARE SIGNIFICANT

* FOR ADDITIONAL INFORMATION, PLEASE CONTACT:

JEFF MANDELBAUM

PRESIDENT, SYBASE FINANCIAL SERVICES, INC.

(+1)510-596-3905

EXAMPLES

* **LIFE CYCLE COSTS**

* **INVESTMENT ANALYSIS**

* **ROI**

ASSUMPTIONS

	SYBASE	DEC R/DB
HARDWARE EFFICIENCY FACTOR	1.00	3.00
DEVELOPMENT PRODUCTIVITY FACTOR	1.00	2.00
MAINTENACE PRODUCTIVITY FACTOR	1.00	2.50
TRAINING PRODUCTIVITY FACTOR	1.00	1.50

PV OF SYBASE ACTUAL COSTS @ 12.5%	$18,926,488
PV OF DEC R/DB ACTUAL COSTS @ 12.5%	$45,942,668

	TODAY	YEAR 1	YEAR 2	YEAR 3	YEAR 4	YEAR 5	TOTAL
RAW COSTS - SYBASE							
HARDWARE	$0	$1,000,000	$2,000,000	$3,000,000	$4,000,000	$0	$10,000,000
SOFTWARE FEES - DEVELOPMENT	$350,000	$0	$0	$0	$0	$0	$350,000
SOFTWARE FEES - RUN-TIME	$800,000	$0	$0	$0	$0	$0	$800,000
SOFTWARE SUPPORT FEES	$0	$110,000	$120,000	$130,000	$140,000	$150,000	$650,000
DEVELOPMENT	$0	$1,000,000	$1,000,000	$1,000,000	$1,000,000	$1,000,000	$5,000,000
MAINTENANCE	$0	$0	$2,000,000	$3,000,000	$4,000,000	$5,000,000	$14,000,000
TRAINING	$0	$80,000	$30,000	$0	$0	$0	$110,000
	$1,150,000	$2,190,000	$5,150,000	$7,130,000	$9,140,000	$6,150,000	$30,910,000
RAW COSTS - R/DB							
HARDWARE	$0	$1,000,000	$2,000,000	$3,000,000	$4,000,000	$0	$10,000,000
SOFTWARE FEES - DEVELOPMENT	$350,000	$0	$0	$0	$0	$0	$350,000
SOFTWARE FEES - RUN-TIME	$0	$0	$0	$0	$0	$0	$0
SOFTWARE SUPPORT FEES	$0	$110,000	$120,000	$130,000	$140,000	$150,000	$650,000
DEVELOPMENT	$0	$1,000,000	$1,000,000	$1,000,000	$1,000,000	$1,000,000	$5,000,000
MAINTENANCE	$0	$0	$2,000,000	$3,000,000	$4,000,000	$5,000,000	$14,000,000
TRAINING	$0	$80,000	$30,000	$0	$0	$0	$110,000
	$350,000	$2,190,000	$5,150,000	$7,130,000	$9,140,000	$6,150,000	$30,110,000
ACTUAL COSTS - SYBASE							
HARDWARE	$0	$1,000,000	$2,000,000	$3,000,000	$4,000,000	$0	$10,000,000
SOFTWARE FEES - DEVELOPMENT	$350,000	$0	$0	$0	$0	$0	$350,000
SOFTWARE FEES - RUN-TIME	$800,000	$0	$0	$0	$0	$0	$800,000
SOFTWARE SUPPORT FEES	$0	$110,000	$120,000	$130,000	$140,000	$150,000	$650,000
DEVELOPMENT	$0	$1,000,000	$1,000,000	$1,000,000	$1,000,000	$1,000,000	$5,000,000
MAINTENANCE	$0	$0	$2,000,000	$3,000,000	$4,000,000	$5,000,000	$14,000,000
TRAINING	$0	$80,000	$30,000	$0	$0	$0	$110,000
	$1,150,000	$2,190,000	$5,150,000	$7,130,000	$9,140,000	$6,150,000	$30,910,000
ACTUAL COSTS - R/DB							
HARDWARE	$0	$3,000,000	$6,000,000	$9,000,000	$12,000,000	$0	$30,000,000
SOFTWARE FEES - DEVELOPMENT	$350,000	$0	$0	$0	$0	$0	$350,000
SOFTWARE FEES - RUN-TIME	$0	$0	$0	$0	$0	$0	$0
SOFTWARE SUPPORT FEES	$0	$110,000	$120,000	$130,000	$140,000	$150,000	$650,000
DEVELOPMENT	$0	$2,000,000	$2,000,000	$2,000,000	$2,000,000	$2,000,000	$10,000,000
MAINTENANCE	$0	$0	$5,000,000	$7,500,000	$10,000,000	$12,500,000	$35,000,000
TRAINING	$0	$120,000	$45,000	$0	$0	$0	$165,000
	$350,000	$5,230,000	$13,165,000	$18,630,000	$24,140,000	$14,650,000	$76,165,000

NOTES:
OUR ASSUMPTIONS ARE DOCUMENTED BY A WIDE VARIETY OF ACTUAL CUSTOMER EXPERIENCES.
INFLATION AND TAXES WERE NOT CONSIDERED IN THIS MODEL.

ASSUMPTIONS								
INITIAL 36 MO. LEASE PAYMENT	76,307							
ESTIMATED FAIR MARKET RESIDUAL	701,993							
AMORITIZATION PERIOD FOR PURCHASE	EXPENSED (TAX SHIELD IS GREATLY REDUCED IF PURCHASES UNDER STANDARD LICENSING ARE CAPITALIZED)							
MARGINAL TAX RATE	40.00%							
ANNUAL PRICE INCREASE (LICENSE FEES)	15.00%							
MAXIMUM SUPPORT INCREASE	10.00%							
CURRENT LICENSE FEE VALUE	9,160,159							
SUPPORT FEES FOR 3 YEAR DEPLOY - STD	1,168,770							
SUPPORT FEES FOR 3 YEAR DEPLOY - OCT	290,652							
CASH FLOWS		OCTOBER '92 PROPOSAL				STANDARD LICENSING		
	LEASE	SUPPORT	TAX	AFTER TAX	LICENSE	SUPPORT	TAX	AFTER TAX
	PAYMENT	FEES	SHIELD	COST	PURCHASE	FEES	SHIELD	COST
Nov-92	76,307	24,221	54,744	21,563	254,449	32,466	134,245	120,203
Dec-92	76,307	24,221	54,744	21,563	254,449	32,466	134,245	120,203
Jan-93	76,307	24,221	54,744	21,563	254,449	32,466	134,245	120,203
Feb-93	76,307	24,221	54,744	21,563	254,449	32,466	134,245	120,203
Mar-93	76,307	24,221	54,744	21,563	254,449	32,466	134,245	120,203
Apr-93	76,307	24,221	54,744	21,563	254,449	32,466	134,245	120,203
May-93	76,307	24,221	54,744	21,563	254,449	32,466	134,245	120,203
Jun-93	76,307	24,221	54,744	21,563	254,449	32,466	134,245	120,203
Jul-93	76,307	24,221	54,744	21,563	254,449	32,466	134,245	120,203
Aug-93	76,307	24,221	54,744	21,563	254,449	32,466	134,245	120,203
Sep-93	76,307	24,221	54,744	21,563	254,449	32,466	134,245	120,203
Oct-93	76,307	24,221	54,744	21,563	254,449	32,466	134,245	120,203
Nov-93	76,307	26,643	57,166	19,141	292,616	53,569	170,615	122,001
Dec-93	76,307	26,643	57,166	19,141	292,616	53,569	170,615	122,001
Jan-94	76,307	26,643	57,166	19,141	292,616	53,569	170,615	122,001
Feb-94	76,307	26,643	57,166	19,141	292,616	53,569	170,615	122,001
Mar-94	76,307	26,643	57,166	19,141	292,616	53,569	170,615	122,001
Apr-94	76,307	26,643	57,166	19,141	292,616	53,569	170,615	122,001
May-94	76,307	26,643	57,166	19,141	292,616	53,569	170,615	122,001
Jun-94	76,307	26,643	57,166	19,141	292,616	53,569	170,615	122,001
Jul-94	76,307	26,643	57,166	19,141	292,616	53,569	170,615	122,001
Aug-94	76,307	26,643	57,166	19,141	292,616	53,569	170,615	122,001
Sep-94	76,307	26,643	57,166	19,141	292,616	53,569	170,615	122,001
Oct-94	76,307	26,643	57,166	19,141	292,616	53,569	170,615	122,001
Nov-94	76,307	29,065	59,588	16,719	330,784	107,137	239,451	91,333
Dec-94	76,307	29,065	59,588	16,719	330,784	107,137	239,451	91,333
Jan-95	76,307	29,065	59,588	16,719	330,784	107,137	239,451	91,333
Feb-95	76,307	29,065	59,588	16,719	330,784	107,137	239,451	91,333
Mar-95	76,307	29,065	59,588	16,719	330,784	107,137	239,451	91,333
Apr-95	76,307	29,065	59,588	16,719	330,784	107,137	239,451	91,333
May-95	76,307	29,065	59,588	16,719	330,784	107,137	239,451	91,333
Jun-95	76,307	29,065	59,588	16,719	330,784	107,137	239,451	91,333
Jul-95	76,307	29,065	59,588	16,719	330,784	107,137	239,451	91,333
Aug-95	76,307	29,065	59,588	16,719	330,784	107,137	239,451	91,333
Sep-95	76,307	29,065	59,588	16,719	330,784	107,137	239,451	91,333
Oct-95	76,307	29,065	59,588	16,719	330,784	107,137	239,451	91,333
Nov-95	701,993	31,487	312,285	389,709	0	139,278	139,278	-139,278
TOTALS	2,747,052	959,152	2,057,972	689,080	10,534,183	2,457,339	6,671,012	3,863,171
PRESENT VALUES								
10.00%	2,881,241	843,890	1,996,386	884,855	8,990,137	2,021,859	5,617,913	3,372,224
15.00%	2,644,568	781,547	1,839,374	805,193	8,332,161	1,839,957	5,172,821	3,159,340
20.00%	2,434,098	725,434	1,699,073	735,025	7,738,812	1,678,090	4,773,615	2,965,197
SAVINGS (PRESENT VALUES)								
10.00%	6,108,896	1,177,969	3,621,527	2,487,369				
15.00%	5,687,593	1,058,410	3,333,447	2,354,146				
20.00%	5,304,713	952,657	3,074,542	2,230,171				
COST PER 4,000 USERS (PRESENT VALUES)								
10.00%				221				843
15.00%				201				790
20.00%				184				741
BREAK EVEN NUMBER OF USERS (PRESENT VALUES)								
10.00%				1,050				
15.00%				1,019				
20.00%				992				

ITEM	TOTAL	INITIAL CASH FLOW	FY 1	FY 2	FY 3	FY 4	FY 5
Hardware Cost	$0	$0	$0	$0	$0	$0	$0
Software Cost	$250,000	$250,000	$0	$0	$0	$0	$0
Sales Taxes @ 7%	$17,500	$17,500	$0	$0	$0	$0	$0
TOTAL	$267,500	$267,500	$0	$0	$0	$0	$0
Income	$862,610	$0	$150,000	$160,500	$171,735	$183,756	$196,619
Expenses	$16,500	$0	$16,500	$0	$0	$0	$0
Lease	$0	$0	$0	$0	$0	$0	$0
Dep. MACRS	$0	$0	$0	$0	$0	$0	$0
Insurance	$0	$0	$0	$0	$0	$0	$0
TOTAL EXPENSES	$16,500	$0	$16,500	$0	$0	$0	$0
Taxable Income	$846,110	$0	$133,500	$160,500	$171,735	$183,756	$196,619
Income Tax at 34%	$287,677	$0	$45,390	$54,570	$58,390	$62,477	$66,850
Net Income	$558,433	$0	$88,110	$105,930	$113,345	$121,279	$129,769
Dep. MACRS	$0	$0	$0	$0	$0	$0	$0
TOTAL INCOME	$558,433	$0	$88,110	$105,930	$113,345	$121,279	$129,769
Cash Flow	$290,933	($267,500)	$88,110	$105,930	$113,345	$121,279	$129,769
Cumulative		($267,500)	($179,390)	($73,460)	$39,885	$161,164	$290,933

Net Present Value at 15% Discount Rate **$97,601**

Internal Rate of Return = **28.53%**

Mainframes &

Connectivity

Presentations

BASF France
DP Department

BASF

AN EXAMPLE OF A COOPERATIVE PROCESS

OVERVIEW

THE CONTEXT

BASF France information systems
55 people
Budget: 60 MF

Base Mainframe:
- **1 COMPAREX 8/90 - MVS CICS DB2**
- **1 COMPAREX 7/75 - VM PROFS AS**
- **310 terminals / 340 PCs**

LANs: 175 workstations at end 93

2 IBM RISC 6000 servers / Databases

THE CONTEXT

- Information technology that is more strategic, of increasing importance in decision-making processes

- Stabilization of costs

- Cooperation within the BASF group (European projects - SAP)

starting point: a culture based on "centralized information technology"

TARGETS

A GLOBAL SOLUTION

- Architecture of operations

and

- Development environment

... as a COMPLEMENT to the existing infrastructure

TARGETS

ARCHITECTURE OF OPERATIONS

- Client/server relational DBMS on UNIX
  Open to DB2

- LAN manager
  Previous requirement: token ring

- Graphic interface on user workstation

TARGETS

DEVELOPMENT ENVIRONMENT

- **Design tool**
  Strong AXIAL culture

- **Source code generator**
  Multi-system (OS/2, W3, UNIX)

THE APPROACH

STRUCTURED

- Definition

- Tests for validation
  PLATFORM ARCHITECTURE TECHNICAL ASPECTS

- Implementation
  PILOT PROJECT
  15 users / application
  30 users / OA equipment on LAN

- Extension
  ---------> 500 users / LAN OA equipment
  300 users / application in 1996

THE APPROACH

PROJECT SCHEDULE

02/91 definition 04/91 tests for validation 01/92

02/92 implementation (pilot project) 07/92 production starts

application
network and operations
development environment

02/93 extension

application
remote networks and portables
administration and tools

TECHNICAL CHOICES

Definition phase

Decision-making criteria for the RDBMS

- **Quality of technical support**
- **Openness (DB2)**
- **Performance levels / security**
- **Functions**
- **Upgradability**

TECHNICAL CHOICES

Definition phase

GRAPHTALK	Rank Xerox / Parallax S.T.
NS-DK	Nat Systèmes
SQL server AIX and gateway DB2	Sybase
NETWARE	Novell
WINDOWS 3	Microsoft

management at a departmental level gets involved

TECHNICAL CHOICES

VALIDATION TESTS PHASE

- Installation and transfer of control
- NS-DK extended to SYBASE
- Definitive generation of DB2 and SYBASE by GRAPHTALK
- Production of "demonstrators"
- Report of experience

the suppliers are integrated into the BASF France project

TECHNICAL CHOICES

CONCLUSION

Main factors of change:

- the graphic interface of the user workstation

- the data distribution

- the heterogeneous "world"

IMPLEMENTATION

PILOT PROJECT

- 40 Screens with graphic interface
  110 days of which 37 days for design

SYBASE Database: 20 MF of which 10 Sales Histories
  (20 tables)

- CICS: 5 read transactions
  DL1 base - Clients
  DB2 tables - Operating reports
  VSAM files - Reference tables

IMPLEMENTATION

PILOT PROJECT

Development, personal computer, network, operations teams brought in alongside the technical support team

... production of application starts end July 1992

EXTENSION

- Commercial plans of action
  20 screens - 50 man days design / implementation
  30 LAN users, 16 client/server users

- Management of non-computer equipment
  20 screens - 50 man days design / implementation
  2 users, to be increased to 10

- Security functionalities
  35 screens - 100 man days design / implementation
  4 users

CONCLUSIONS

1 - Quantitive elements

2 - What was gained

3 - Main lessons learned

CONCLUSIONS

QUANTITIVE ELEMENTS

- Definition phase = 60 man days / spread over 1.5 months

- Validation phase on platform:

  1 million French francs invested

  260 man days of which:
  60% installation / transfer of control
  18% development of demonstrators
  15% reporting on experience
  7% training

CONCLUSIONS

QUANTITIVE ELEMENTS

- Implementation phase / pilot project

  360 man days of which:

  58% network and operations
  28% application
  7% development environment (definition)
  7% training

Total time of project = 18 months

CONCLUSIONS

WHAT WAS GAINED

- Effective use of existing PC power
- Reduction of development costs

  factor 4 observed
- Operating costs
- User involvement

CONCLUSIONS

MAIN LESSONS LEARNED

- the solution is of value

- it is realistic to look for a complete solution

OPERATIONAL IN 18 MONTHS

- the problems are real

- the cultural leap is significant

BASF France
Département Informatique

By Gisele QUANTIN FERNAUD

UN EXEMPLE DE

PROCESSUS COOPERATIF

BASF

réf: syb10 - 19/07/93

BASF

PLAN

1 - Présentation de BASF

2 - Le contexte

3 - Objectifs

4 - La démarche

5 - Choix techniques

6 - Mise en oeuvre

7 - Extension

8 - Conclusions

9 - Questions / Réponses

LE CONTEXTE

Systèmes d'information de BASF France

55 personnes

Budget de 60 MF

Base Mainframe :

1 COMPAREX 8/90 - MVS CICS DB2

1 COMPAREX 7/75 - VM PROFS AS

310 terminaux / 340 micros

Réseaux locaux : 175 postes à fin 93

2 Serveurs IBM RISC 6000 / Bases de données

LE CONTEXTE

- Informatique plus stratégique et décisionnelle
- Stabilisation des coûts
- Coopération au sein du groupe BASF
  (projets européens - SAP)

au départ : une culture "Informatique centralisée"

OBJECTIFS

UNE SOLUTION GLOBALE

- **Architecture d'exploitation**

et

- **Environnement de développement**

... en COMPLEMENT de l'infrastructure existante

OBJECTIFS

ARCHITECTURE D'EXPLOITATION

- SGDB relationnel client-serveur sur UNIX

  Ouvert sur DB2

- Gestionnaire de réseau local

  Préalable : token ring

- Interface graphique sur poste de travail utilisateur

OBJECTIFS

ENVIRONNEMENT DE DEVELOPPEMENT

- Outil de conception

  Forte culture AXIAL

- Générateur de code source

  Multi cibles (OS/2, W3, UNIX)

LA DEMARCHE

STRUCTUREE

- **Définition**

- **Validation par expérimentation**

  PLATE-FORME ARCHITECTURE TECHNIQUE

- **Mise en oeuvre**

  PROJET PILOTE

  15 utilisateurs / applicatif

  30 utilisateurs / bureautique en réseau local

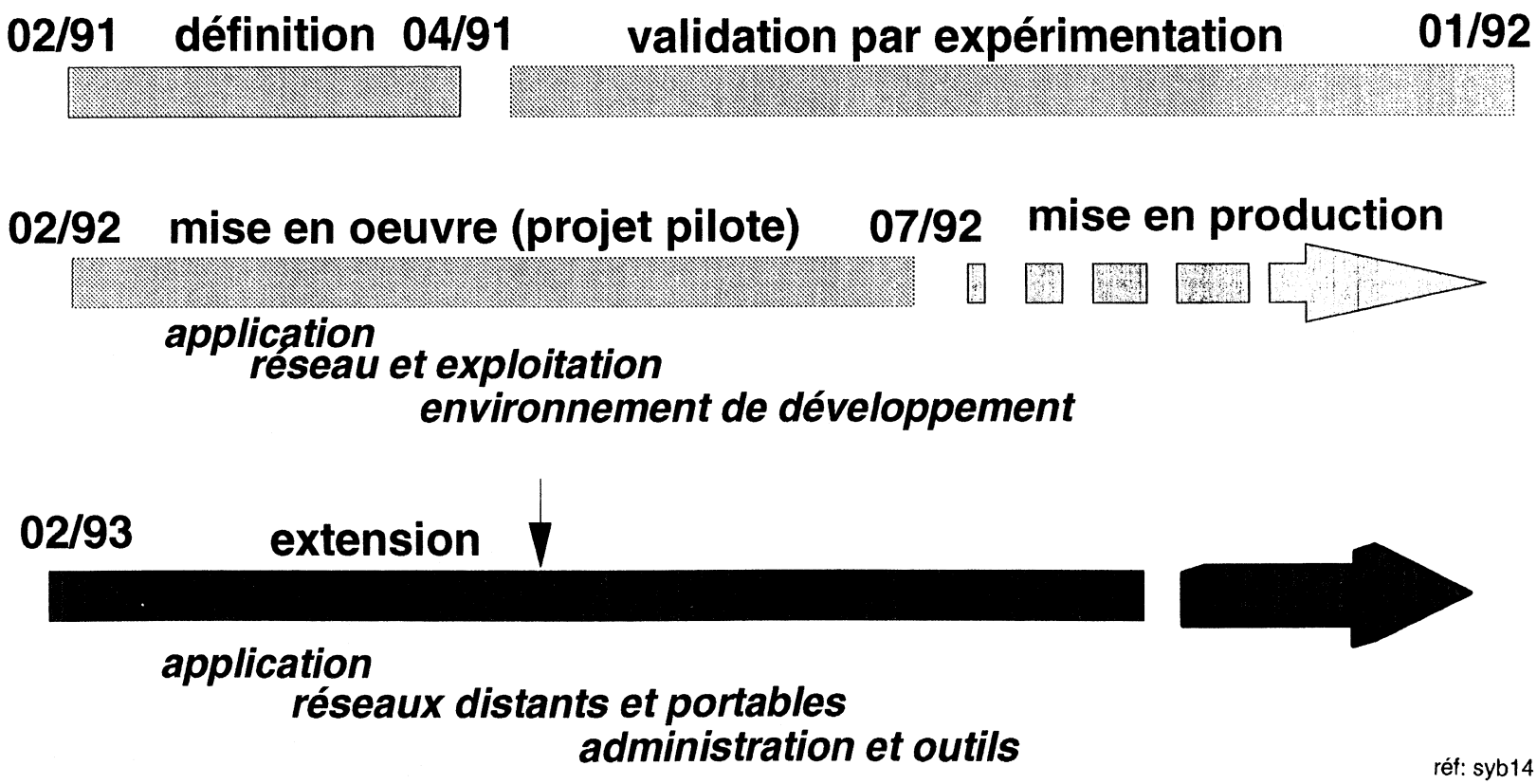
- **Extension** → **500 utilisateurs / bureautique RL**
  300 utilisateurs / applicatif en 1996

réf: syb13 - 19/07/93

BASF

LA DEMARCHE

PLANNING GENERAL



réf: syb14 - 19/07/93

BASF

CHOIX TECHNIQUES

Phase de définition

Critères de choix pour le SGBDR

- Qualité du support technique
- Ouverture (DB2)
- Performances / sécurité
- Fonctionnalités
- Perspectives d'évolution

CHOIX TECHNIQUES

Phase de définition

GRAPHTALK	Rank Xerox / Parallax S.T
NS-DK	Nat Systèmes
SQL server AIX et gateway DB2	Sybase
NETWARE	Novell
WINDOWS 3	Microsoft

la hiérarchie du département s'implique

réf: syb04 - 03/12/92

BASF

CHOIX TECHNIQUES

Phase de validation par expérimentation

- Installation et prise en main
- Extension de NS-DK à SYBASE
- Génération définit. DB2 et SYBASE par GRAPHTALK
- Réalisation de " démonstrateurs "
- Rapport d' expérience

les fournisseurs sont intégrés au projet de BASF France

CHOIX TECHNIQUES

CONCLUSION

Principaux facteurs de changement :

- l'interface graphique du poste utilisateur

- la distribution des données

- le "monde" hétérogène

MISE EN OEUVRE

PROJET PILOTE

- 40 Ecrans interface graphique
  110j dont 37j en conception

- Base de données SYBASE : 20M dont 10 Histo. Ventes
  (20 tables)

- CICS : 5 transactions en lecture
  Base DL1 - Clients
  Tables DB2 - Tableaux de bord
  Fichiers VSAM - Tables de référence

MISE EN OEUVRE

PROJET PILOTE

implication des équipes développement,
micro-informatique, réseau, exploitation,
aux côtés de l'équipe support technique.

... passage en production de l'applicatif à fin juillet 92.

EXTENSION

- Plans d'action commerciaux
  20 écrans - 50 hj conception / réalisation
  30 utilisateurs réseau local, 16 client/serveur

- Gestion matériel non informatique
  20 écrans - 50 hj conception / réalisation
  2 utilisateurs, évolution 10

- Fiches de sécurité
  35 écrans - 100 hj conception / réalisation
  4 utilisateurs

réf: syb17 - 19/07/93

BASF

CONCLUSIONS

1 - Eléments quantitatifs

2 - Bénéfices induits

3 - Principaux enseignements

CONCLUSIONS

ELEMENTS QUANTITATIFS

- Phase de définition = 60 hj / durée 1,5 mois
- Phase de validation sur plateforme :

1 MF d'investissements

260 hj dont :

60 % installation / prise en main
18 % dév. démonstrateurs
15 % rapport expérience
7 % formation

CONCLUSIONS

ELEMENTS QUANTITATIFS

- Phase de mise en oeuvre / projet pilote

360 hj dont :

58 % réseau et exploitation

28 % applicatif

7 % env de développement (définition)

7 % formation

Durée totale du projet = 18 mois

CONCLUSIONS

BENEFICES INDUITS

- Utilisation effective de la puissance PC installée
- Charge de développement réduite

  facteur 4 constaté

- Coût d'exploitation
- Utilisateurs intéressés

CONCLUSIONS

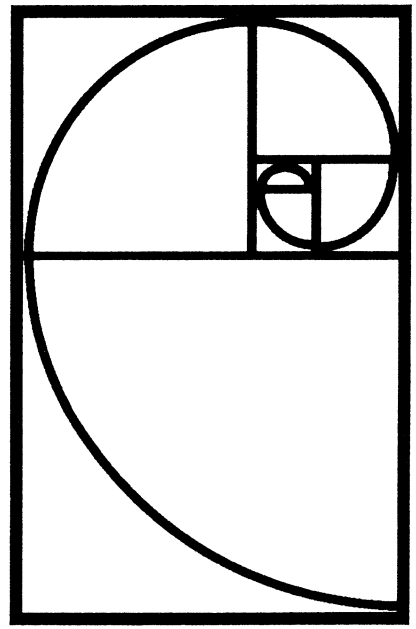
PRINCIPAUX ENSEIGNEMENTS

- l'intérêt de la solution
- la recherche d'une solution complète est réaliste

opérationnelle en 18 mois

- les difficultés sont réelles
- le saut culturel est important

Integrating SYBASE Systems for a PC



SYBASE®

Client/Server Architecture for the On-Line Enterprise

By Dr. Wolfgang Martin

Topics

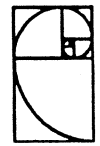
The Challenge of Enterprise Client/Server

The Right Architecture

SYBASE Advances the State of the Art

Enterprise Client/Server Successes

Summary



Topics

The Challenge of Enterprise Client/Server

The Right Architecture

SYBASE Advances the State of the Art

Enterprise Client/Server Successes

Summary

MFCS.2

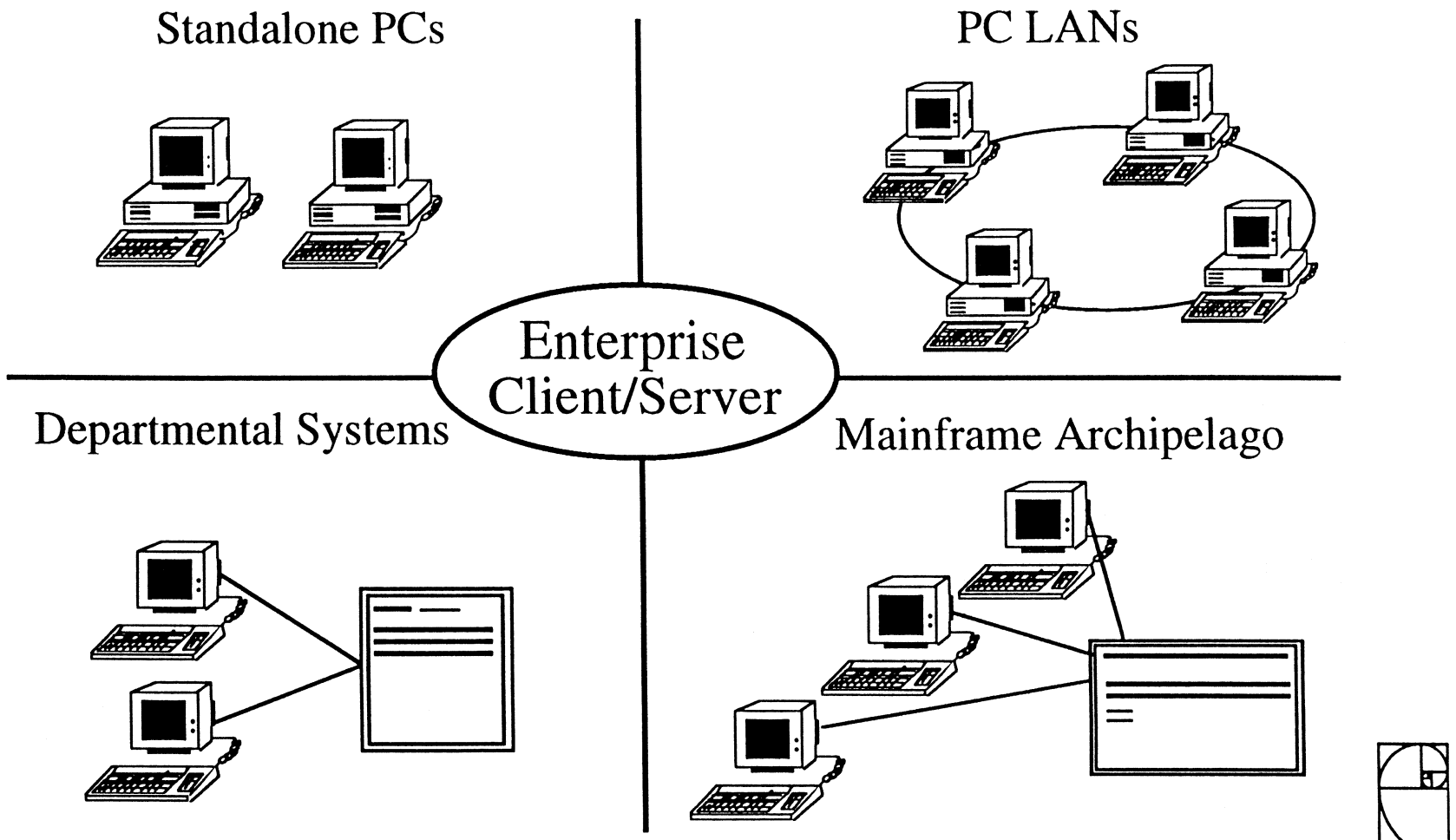
Audience:

This presentation is designed to provide details of the SYBASE Open Interoperability story to the Director of MIS, Managers of Information Systems and specific project leaders.

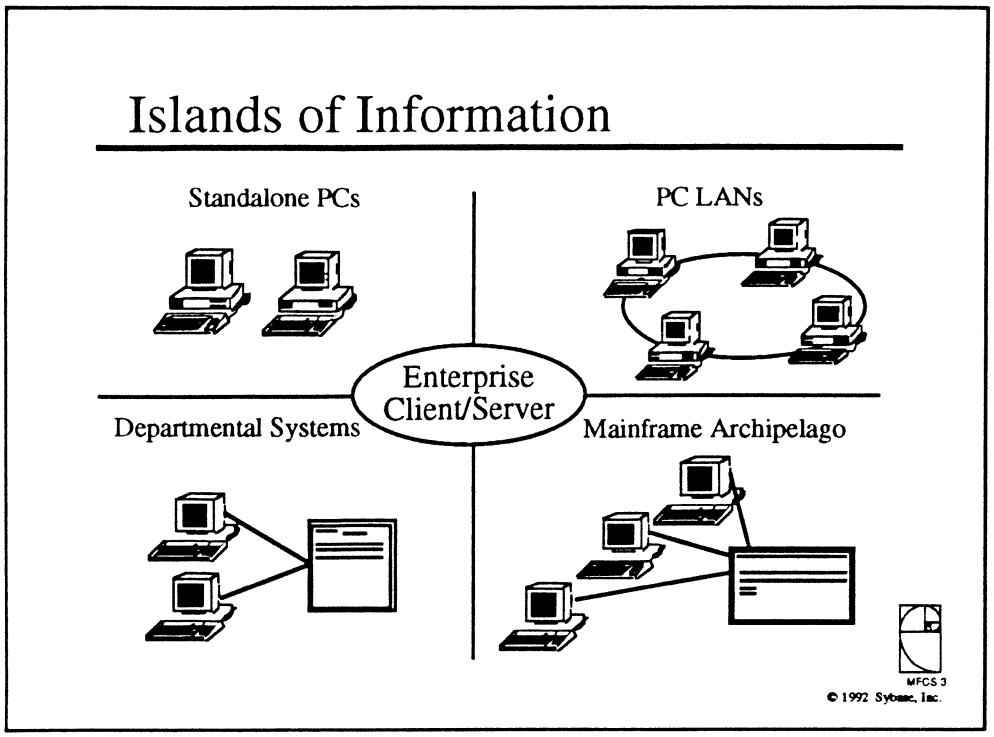
Key Messages:

- SYBASE Client/Server Interfaces will make all your information resources available anywhere in your organization
- SYBASE enables you to fully leverage your current hardware and software investments
- SYBASE enables you to map your information systems to your business requirements rather than retrofitting business requirements onto an inflexible architecture
- Sybase Interoperability strategy is a dual strategy with toolkits for building interoperable products and turnkey gateways for transparent access to data sources
- Sybase Client/Server "glue" reduces the work you must do and makes your job easier

Islands of Information



MFCS.3



Key Points:

- Most corporate enterprises consist of diverse islands of information
- Inability to share information between workgroups leads to inaccurate or untimely decisions and **costs money**

Notes:

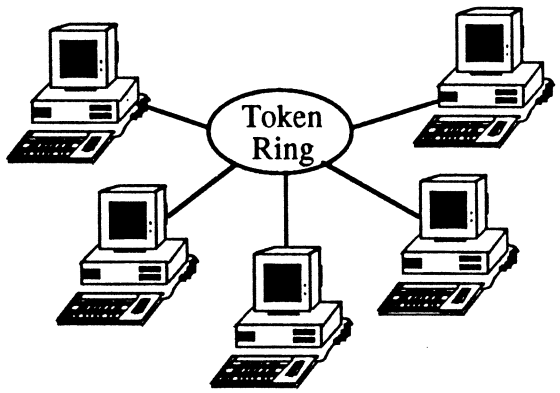
The emergence of departmental and workgroup computing has enabled end-users to define and develop individual and workgroup applications and data sources. These applications and data sources have become a valuable asset to the corporation.

Unfortunately, the potential of new information technologies has gone unrealized because the different systems throughout the enterprise cannot communicate. Workstation users are unable to access key corporate data on the central mainframe. Mainframe users cannot access departmental data on the LAN.

As a result, companies make decisions based on untimely or inaccurate data. The results are lost opportunities and expensive errors.

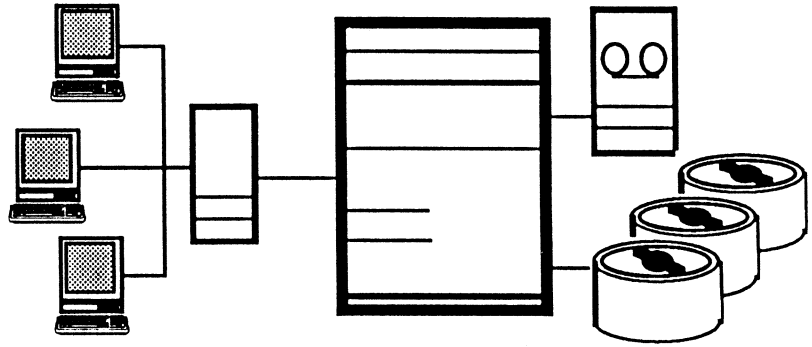
Leverage Current Strengths

Workgroup Computing



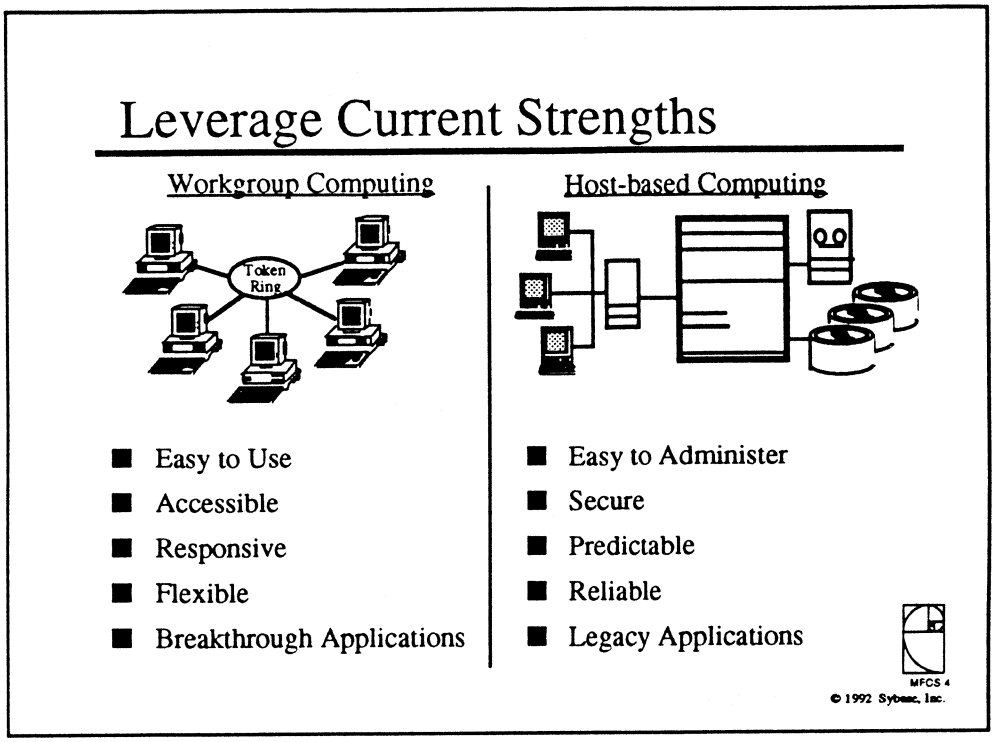
- Easy to Use
- Accessible
- Responsive
- Flexible
- Breakthrough Applications

Host-based Computing



- Easy to Administer
- Secure
- Predictable
- Reliable
- Legacy Applications

MFCS.4



Key Points:

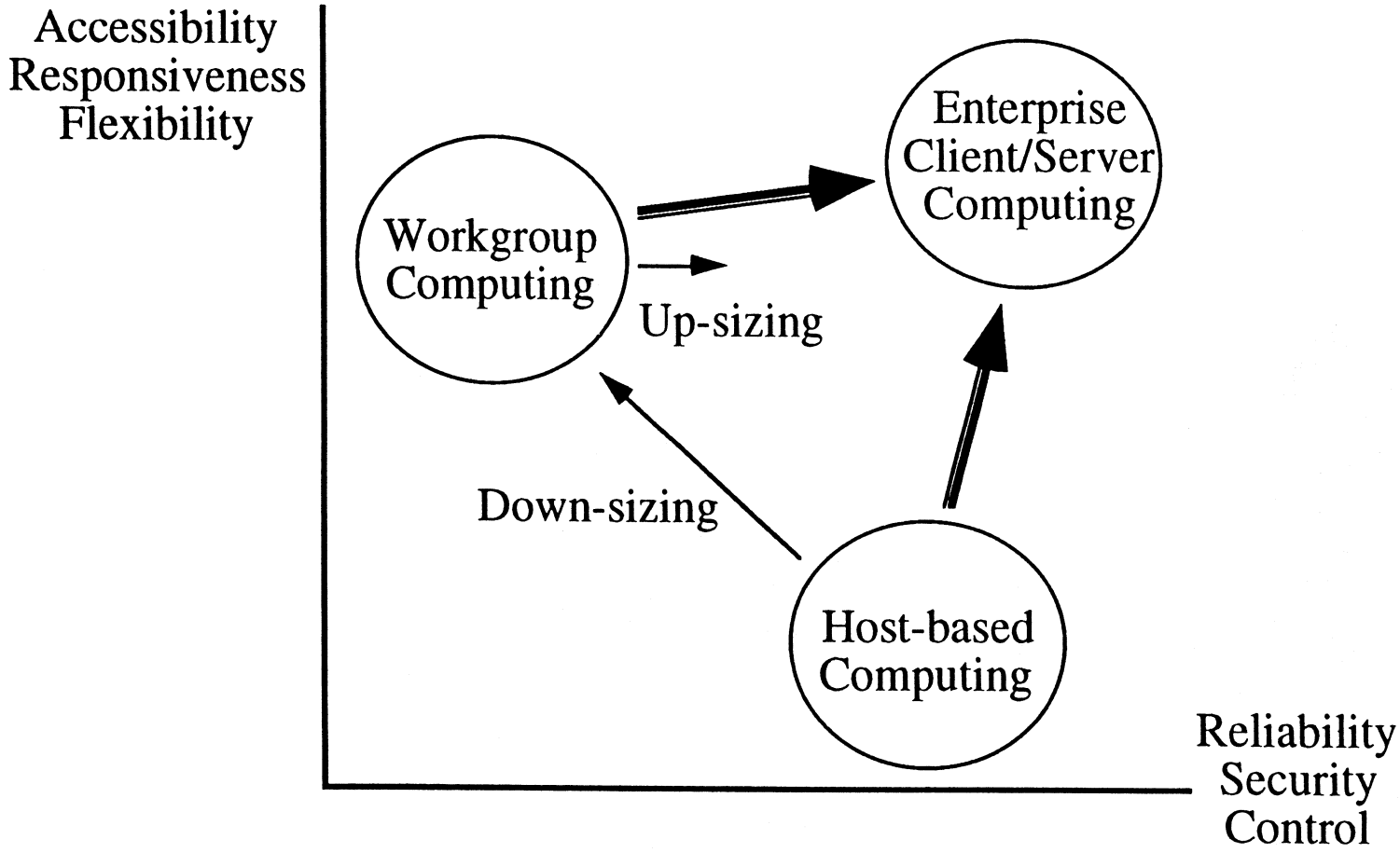
- Your goal is to leverage the strengths of your current systems, not to replace one with the other
- This requires balancing the free access of the desktop and the security and integrity of the host environment

Notes:

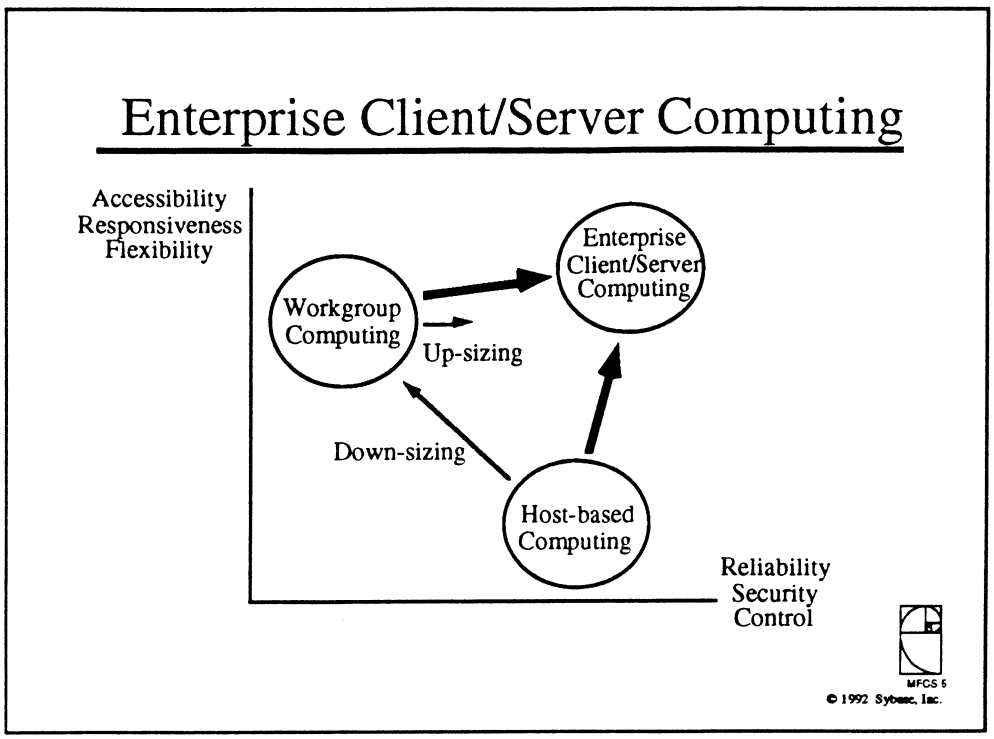
As you attempt to unify the islands of information within your organization, you must balance the freedom of the desktop with the control of the host environment. By leveraging the strengths of your existing systems, you can extend the flexibility and responsiveness typical of workgroup systems to mainframe data. Without the loss of control over the security, integrity and reliability of that key corporate resource.

Of course, if you can leverage your investment in current systems (by re-using them, rather than re-writing them), you will also save money during the transition.

Enterprise Client/Server Computing



MFCS.5



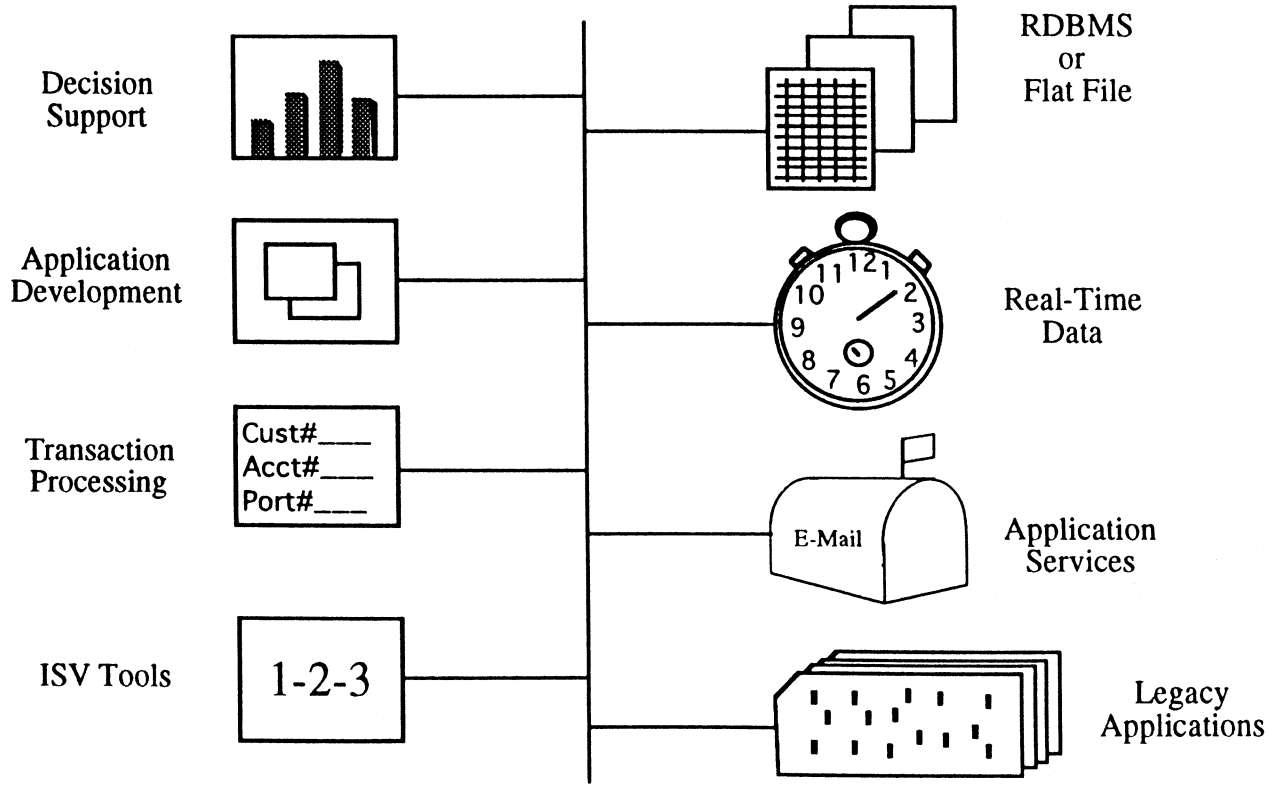
Key Points:

- SYBASE Enterprise Client/Server Computing is the way to unify the islands of information throughout the company
- Enterprise Client/Server combines the benefits of Workgroup and Host-based computing into a whole greater than the sum of the parts

Notes:

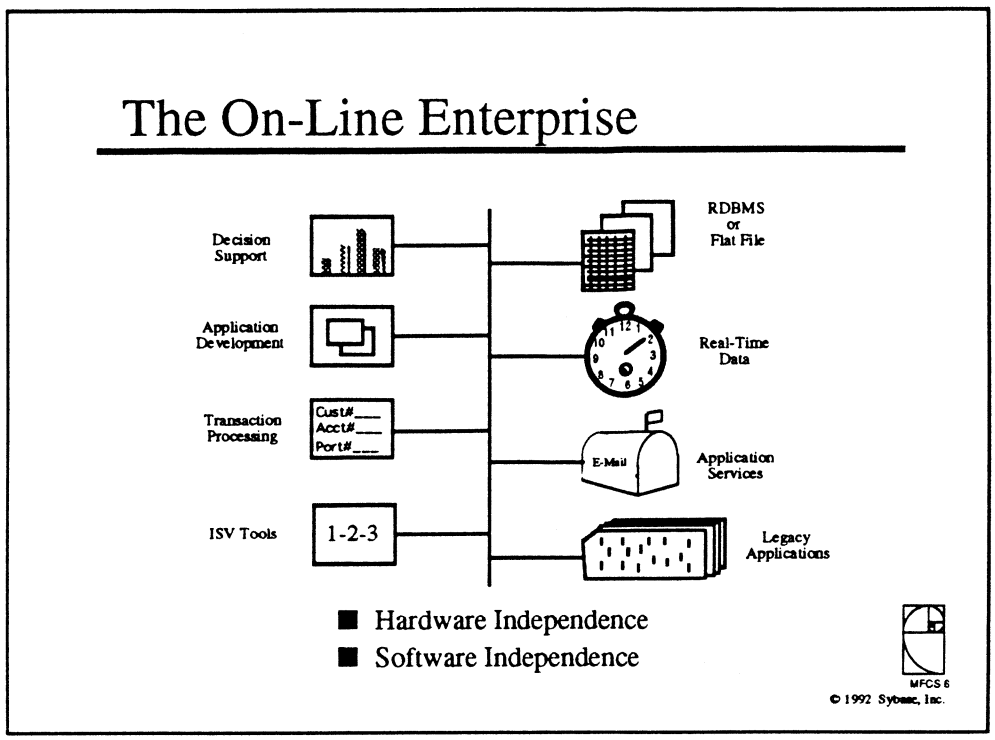
- Simply moving your information operations from one model to the other is not the answer. Instead, you need an architecture which combines the strengths of both models.
- "Downsizing" simply moves host-based systems to a smaller platform where they will meet the needs of a smaller workgroup.
- "Upsizing" moves some of a workgroup's data to a LAN-based server.
- Neither approach addresses the real problem: the inability to share data and applications between workgroups (a benefit of the host-based environment) while providing the flexibility and responsiveness typical of desktop systems. Only Enterprise Client/Server can do that.

The On-Line Enterprise



- Hardware Independence
- Software Independence

MFCS.6



Key points

- Client and Server go beyond just SQL applications and SQL Databases

- Many enterprise data sources include SQL databases along side non-SQL data sources and real-time data which will never be replaced by any static database.
  - SQL databases (SYBASE, Oracle, DB2)
  - Non-SQL data sources (DBase, RMS, text files)
  - Real-time data feeds (stock feeds, process control data)
  - System Services (E-Mail, X.500 Directory Services)
  - Legacy applications (Mainframe CICS applications, VSAM, old mainframe COBOL applications)

- You need more than just platform portability or data gateways. You need true application interoperability: the ability to freely combine client and server applications into complete business solutions.

- Oracle defines Open Systems as the ability to run Oracle on any platform or network. Sybase defines Open Systems as the freedom to choose to combine applications and data from multiple hardware and software vendors to solve your business problem.

Topics

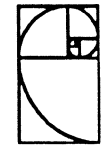
The Challenge of Enterprise Client/Server

The Right Architecture

SYBASE Advances the State of the Art

Enterprise Client/Server Successes

Summary



Topics

The Challenge of Enterprise Client/Server

The Right Architecture

SYBASE Advances the State of the Art

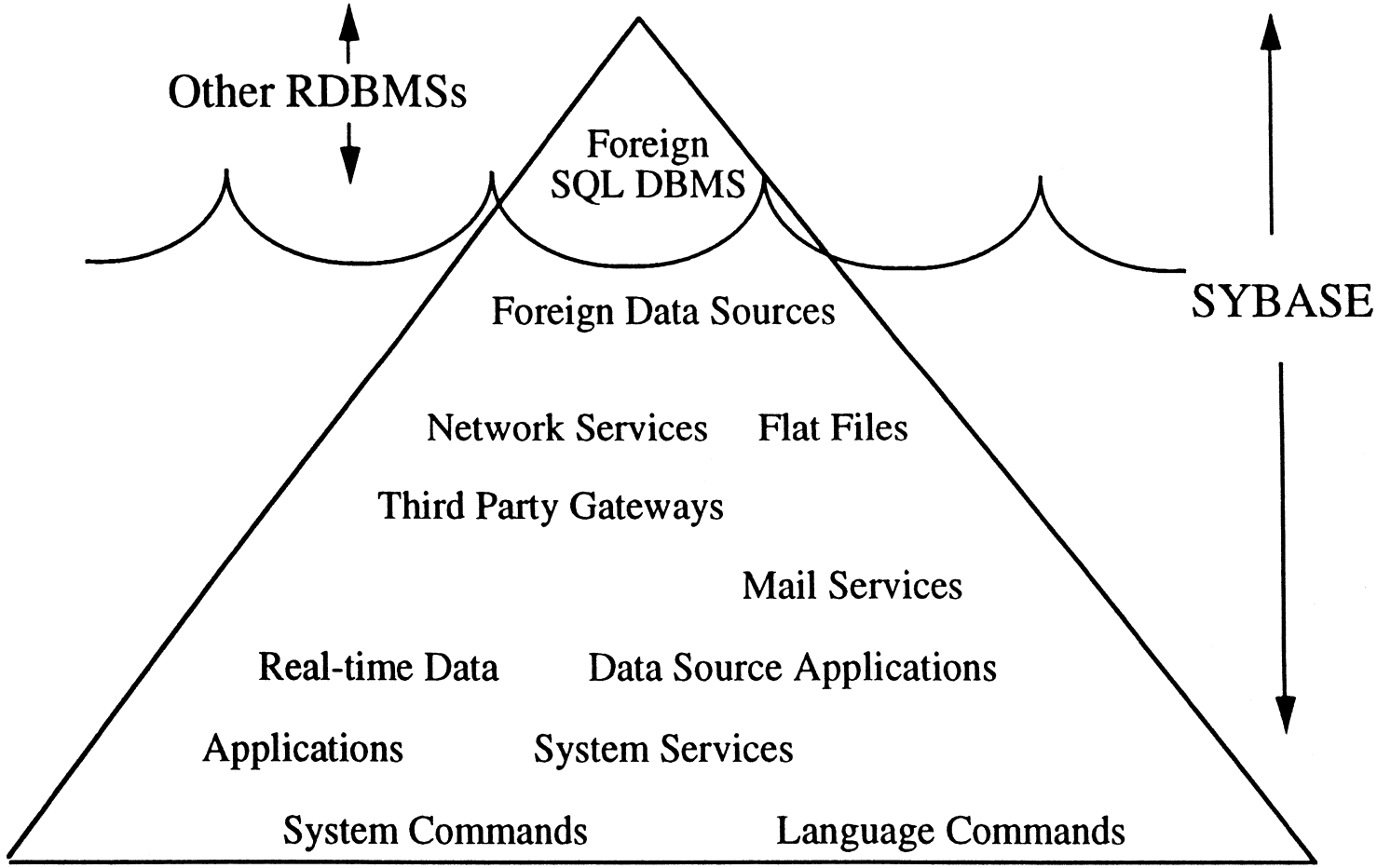
Enterprise Client/Server Successes

Summary

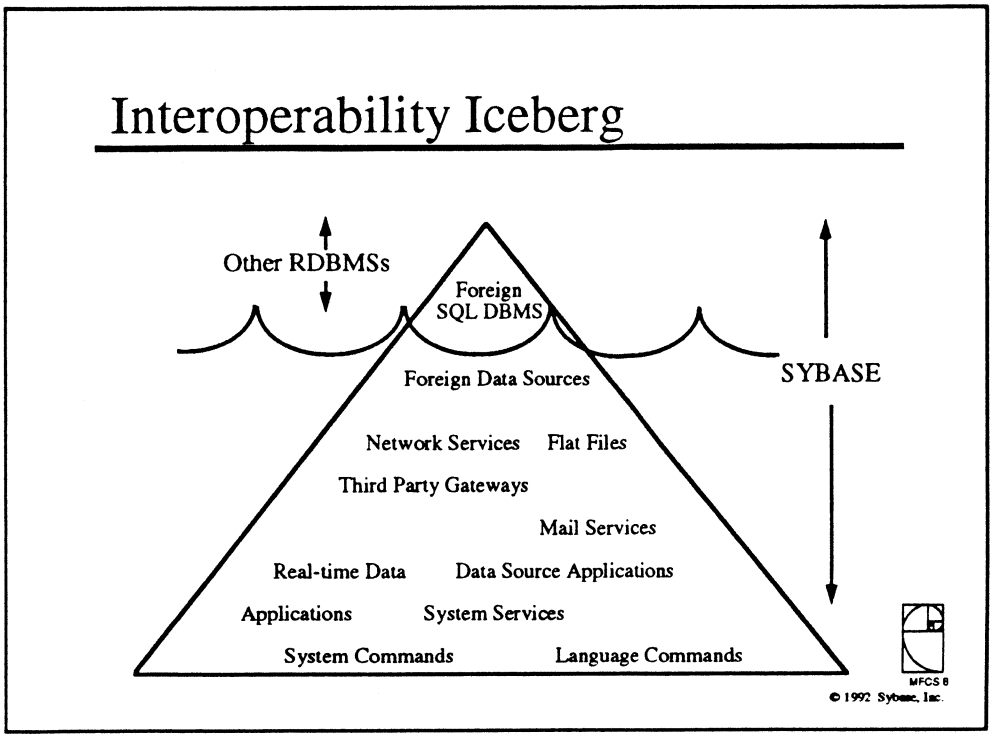
MFCS 7

Let's take a look at what it takes to deliver a truly enterprise-wide interoperability solution and define an architecture that can meet these needs.

Interoperability Iceberg



MFCS.8



Key Points:

- Your customer may be looking at interoperability as a SQL database interoperability issue. Sybase views this as a very important problem to solve, however it's only the tip of the iceberg.
- SYBASE provides the toolkits needed to address all the enterprise interoperability requirements including SQL databases and ANY other data source.

Notes:

Most RDBMS vendors talk about SQL-based gateways when they should be talking about true application interoperability. SQL gateways begin to address your enterprise interoperability problem, but they're only the tip of the iceberg.

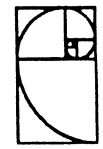
For true enterprise-wide interoperability, you need to address the full spectrum of data and applications spread throughout your enterprise. You need to deal with SQL and non-SQL data, with application-to-application communications, with systems from different hardware and software vendors and still maintain your current standards for security and performance.

You need SYBASE.

The Right Architecture

- Rationalizes Multi-Vendor Chaos
- Maps IS Structure to Business Organization
- Reduces Programming Costs and Application Backlog
- Balances Flexibility and Reliability
- Improves Performance and Availability
- Integrates New and Existing Technology

Simplifies Your Business Operations



MFCS.9

The Right Architecture

- Rationalizes Multi-Vendor Chaos
- Maps IS Structure to Business Organization
- Reduces Programming Costs and Application Backlog
- Balances Flexibility and Reliability
- Improves Performance and Availability
- Integrates New and Existing Technology

Simplifies Your Business Operations

MFCS 9

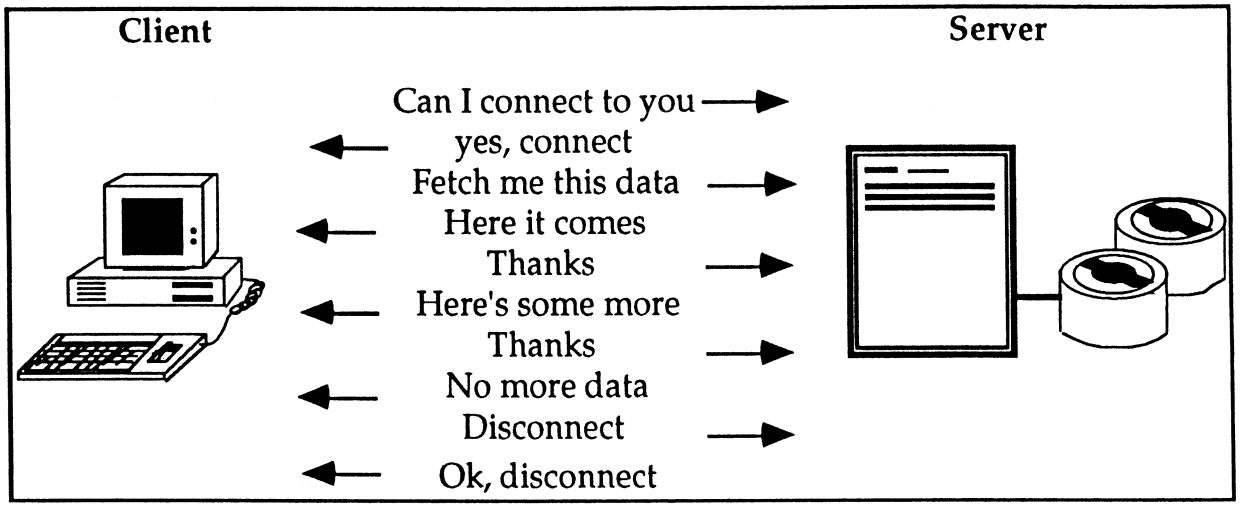
Key Points:

- The challenge is to find the right architecture.
- The right architecture is the one that:
  - Simplifies Your Business Operations
  - Makes You More Competitive
  - Leverages Your Current Personnel and System Resources
  - Enables You to Spend Time Increasing Profits
  - Reduces Your Applications Backlog

Notes:

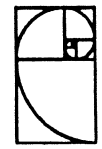
The right architecture is one which allows your to unify your business requirements with your current system. It must also make programming easier and faster, thus reducing costs and application development backlog. It must be able to balance the integration of new technology and applications while ensuring security and control.

Client/Server Conversations



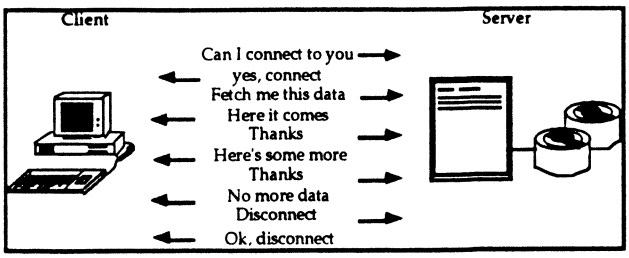
Issues:

- Application Conversation Flows
- Data Conversions
- Error Handling
- Security
- Networking



MFCS.10

Client/Server Conversations



Issues:

- Application Conversation Flows
- Data Conversions
- Error Handling
- Security
- Networking

MFCS 10

Notes:

What does client/server application communication require? The key to enabling transparent client/server integration is application interoperability.

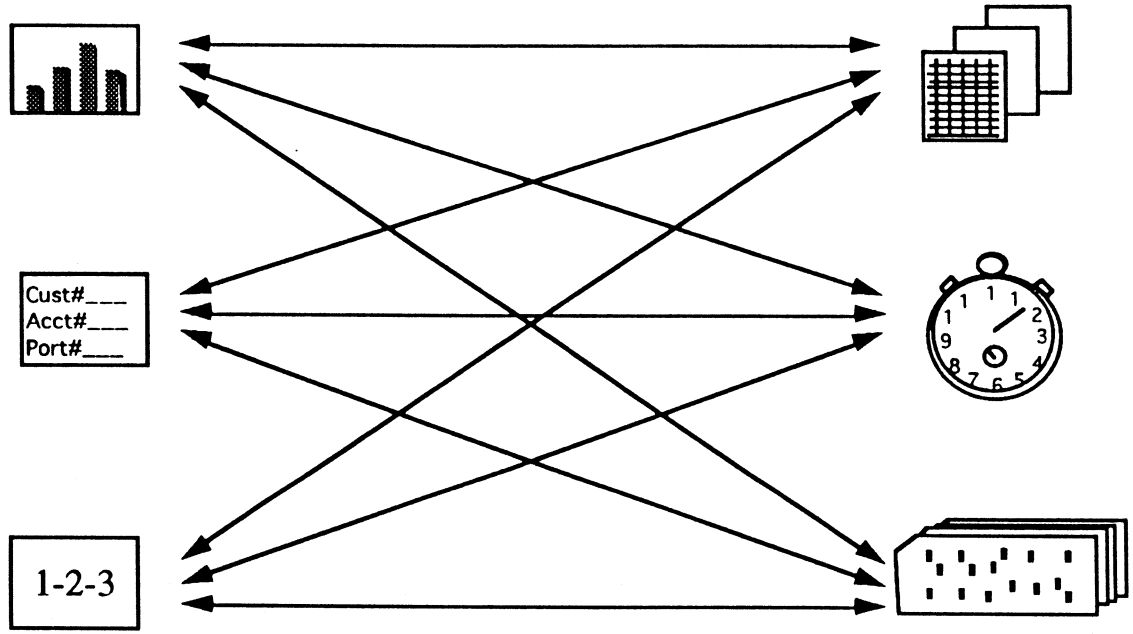
- Do both applications support the same physical network connection?
- Do both applications support the same network protocol?
- Do both applications support the same application protocol?
- Do both applications understand the same commands and syntax?

The first two issues are resolved by the networking communication software and hardware. The second two issues, which relate to application communication rules and syntax, are more complex.

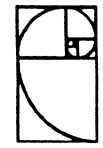
An application protocol is a set of rules defining the interaction between applications on a network. Much like a telephone conversation, in which "hello, may I speak to Bob Jones" is the "implied" rule (or protocol) for requesting to speak to another person and "Good bye, Bob" is the protocol for ending a conversation, the application layer contains a protocol for communication of application messages.

The command language is the second issue applications need to resolve. Both applications must understand the same command set. In databases, the command language is SQL. For non-database applications the language commands can be anything from RPCs to language commands.

What's Wrong with this Picture?

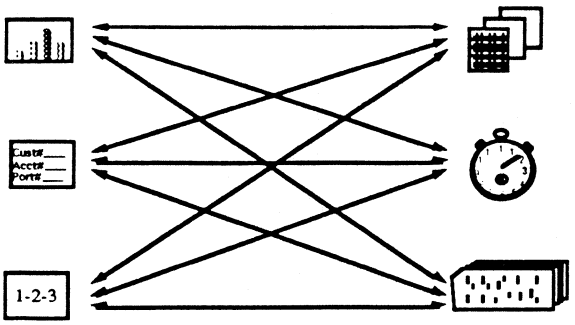


- High Development Costs
- Separate Development Effort for Each Pair of Applications
- Increased Application and Network Maintenance
- Difficult to Guarantee Integrity and Security



MFCS.11

What's Wrong with this Picture?



- High Development Costs
- Separate Development Effort for Each Pair of Applications
- Increased Application and Network Maintenance
- Difficult to Guarantee Integrity and Security

MFCS 11

Key points

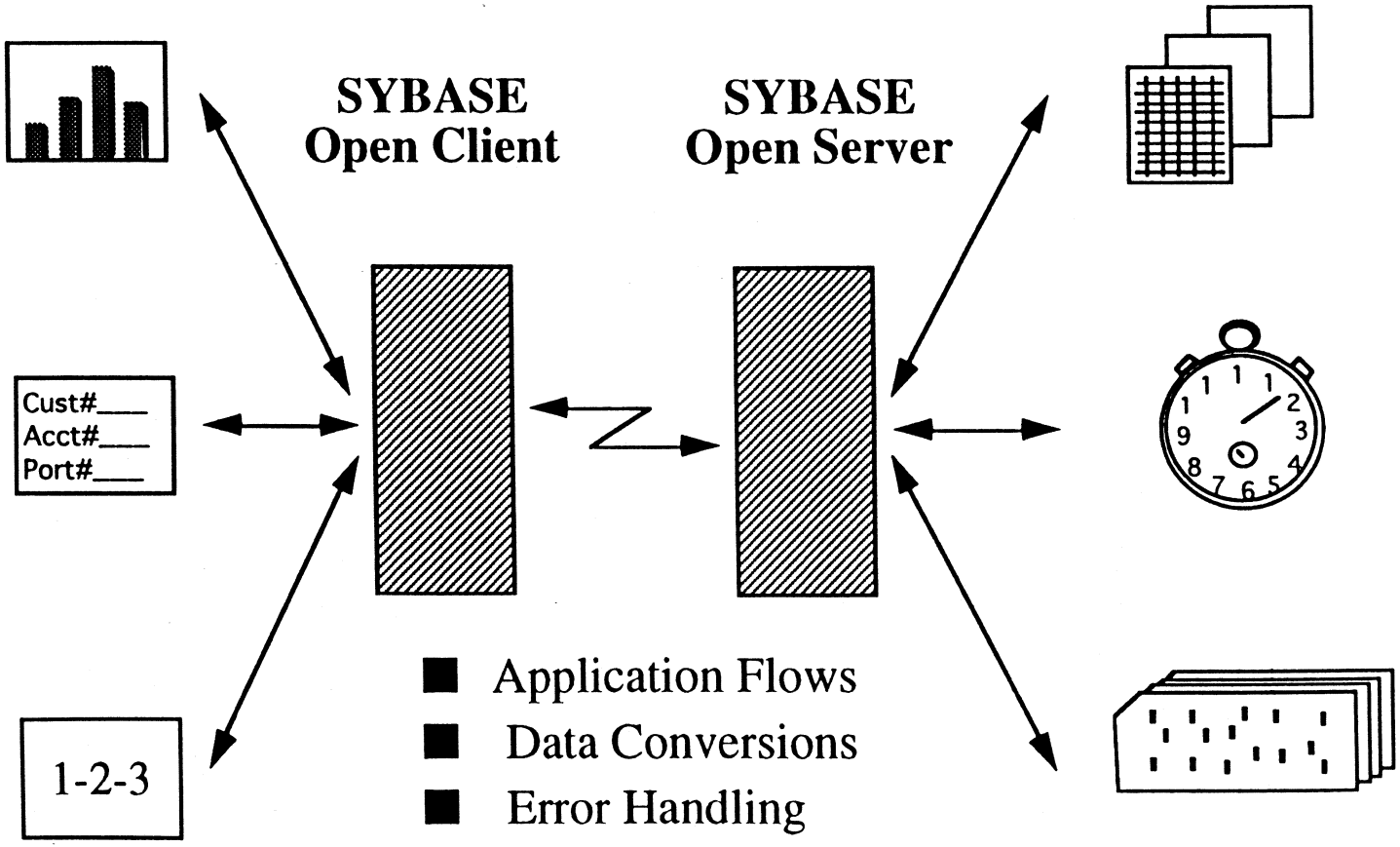
- With a do it yourself connectivity solution every application-to-application connection is a special case
- Development and maintenance costs are high
- Corporate growth is limited by the ability of MIS to keep up with required new connections and applications

Notes:

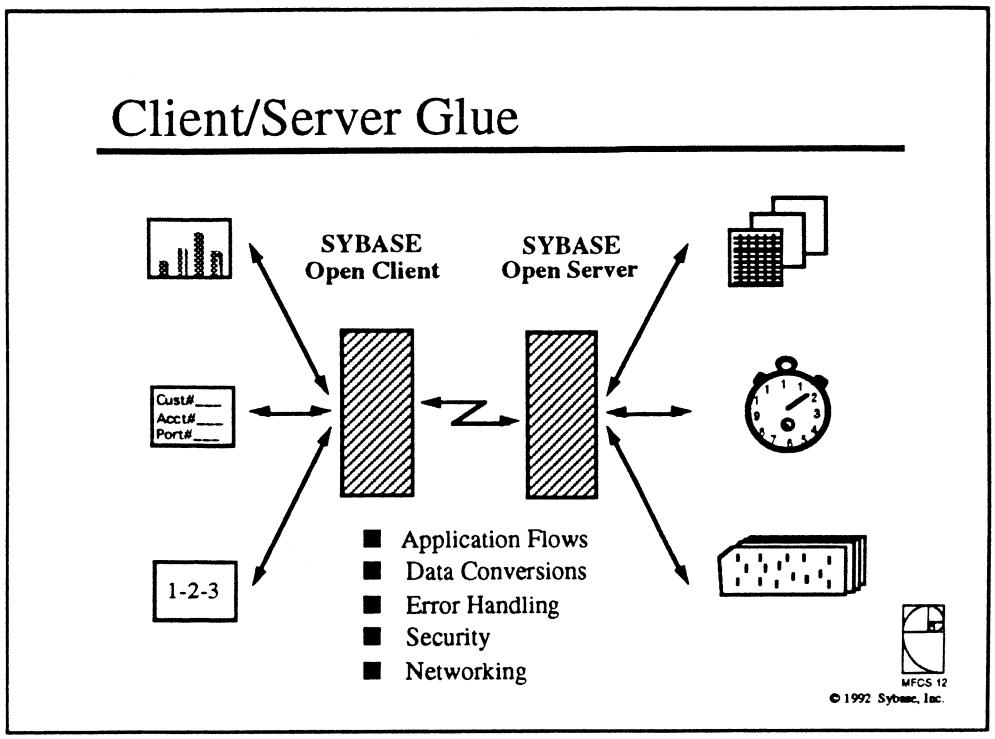
There are connectivity tools that will enable programmers to connect an application on the network to a data source (some of them we mentioned already). However, the problem is that every application-to-data source or application-to-application connection is a separate and special connection. For example the commands required to connect Lotus 1-2-3 to a flat file system such as ISAM is very different than to have Lotus 1-2-3 access stock feed data from a stock feed. Each time, special code needs to be written on both the client-side and server-side.

This results in high development and maintenance costs, since each connection and each new version of the data source or application is a special case. It is difficult to ensure transactional or data integrity since there is no system security or transaction model. Corporate growth is limited by the ability to write and maintain each connection. The result can be major MIS application backlog .

Client/Server Glue



MFCS.12



Key Points:

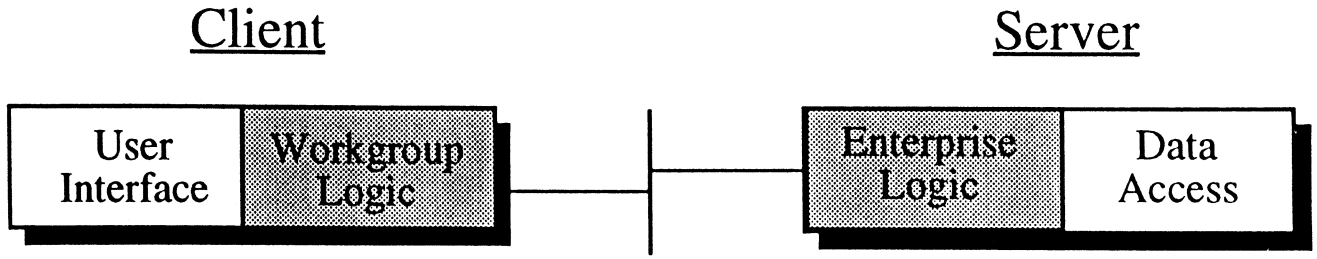
- To make connecting disparate clients and servers an easy task to develop and maintain, programmers need a common glue
- Open Client and Open Server are toolkits for developing client and server applications without the complexity of writing networked based applications

Notes:

The solution is to glue your client and server applications, and data sources, with common interoperability toolkits. SYBASE Open Client and Open Server provide this connectivity glue, allowing client applications to make data requests as if the data were resident on the client system and allowing servers to respond to client requests as if they were made from within the server.

SYBASE Open Client and Open Server provide developer services for programming client and server applications. In addition, they provide runtime services which shield the complexities of developing networked client/server applications from the application developer. This eliminates the special case problem. Over 600 ISVs and VARs support Sybase's Open Client interfaces, and there are a number of interfaces to key databases. Additionally, third parties and customers can easily integrate their own data sources without the burden of developing network based applications.

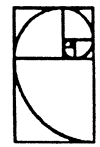
SYBASE Enterprise Client/Server



- Easy to Use
- Accessible
- Responsive
- Flexible
- Breakthrough Applications

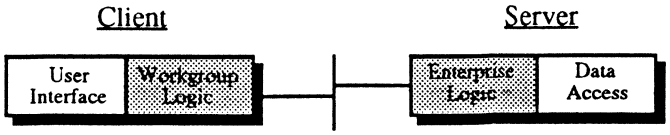
- Easy to Administer
- Secure
- Predictable
- Reliable
- Legacy Applications

Leverages the Strengths of Your Existing Systems



MFCS.13

SYBASE Enterprise Client/Server



- Easy to Use
- Accessible
- Responsive
- Flexible
- Breakthrough Applications

- Easy to Administer
- Secure
- Predictable
- Reliable
- Legacy Applications

Leverages the Strengths of Your Existing Systems

MFCS 13

Key Points:

• Open Client and Open Server enable you to leverage the strengths of your computing environment with flexibility for users and programmers and control and performance from the server.

Notes:

Remembering slide 4, one of our goals is to provide solutions that leverage the strengths of your existing systems. With the SYBASE Open Client and Open Sever products, users get the responsiveness and flexibility they need while the data is accessible yet secure. This architecture allows you to integrate new technology and leverage your legacy technology by allowing you to freely access it from within your enterprise.

With this clean, modular architecture, users and programmers experience the balance between flexibility and control.

Topics

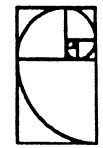
The Challenge of Enterprise Client/Server

The Right Architecture

SYBASE Advances the State of the Art

Enterprise Client/Server Successes

Summary



Topics

The Challenge of Enterprise Client/Server

The Right Architecture

SYBASE Advances the State of the Art

Enterprise Client/Server Successes

Summary

MFCS 14

Notes:

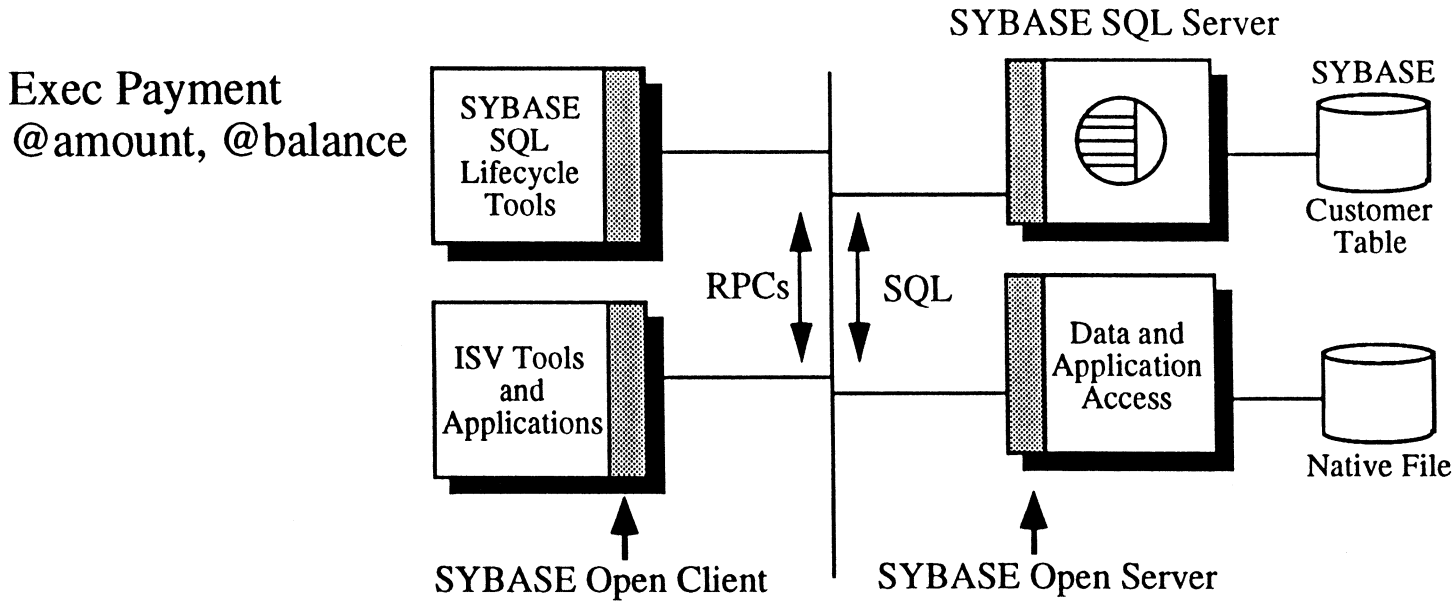
Now we are going to talk about our products and how they can help solve your business and enterprise requirements.

Note that Sybase has a dual interoperability strategy:

- to provide open interoperability products for the integration of ANY data source or application
- to provide gateways for transparent turnkey access to key databases.

It is with this dual strategy that Sybase can help you integrate your entire enterprise with maximum flexibility, ease-of-use and control.

Balances Flexibility and Control

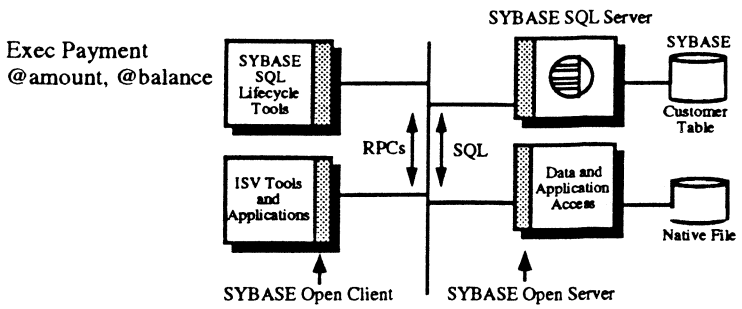


SYBASE Client/Server Interfaces Provide

- Choice of Tools and Data Sources (Realtime, Flat Files)
- Re-Use of Legacy Applications as Servers
- Full Access Control through RPCs

MFCS.15

Balances Flexibility and Control



SYBASE Client/Server Interfaces Provide

- Choice of Tools and Data Sources (Realtime, Flat Files)
- Re-Use of Legacy Applications as Servers
- Full Access Control through RPCs

Notes:

SYBASE Open Client is a toolkit for building client applications. It provides developer and runtime services for creating distributed applications that request information and services from network applications. Open Client supports:

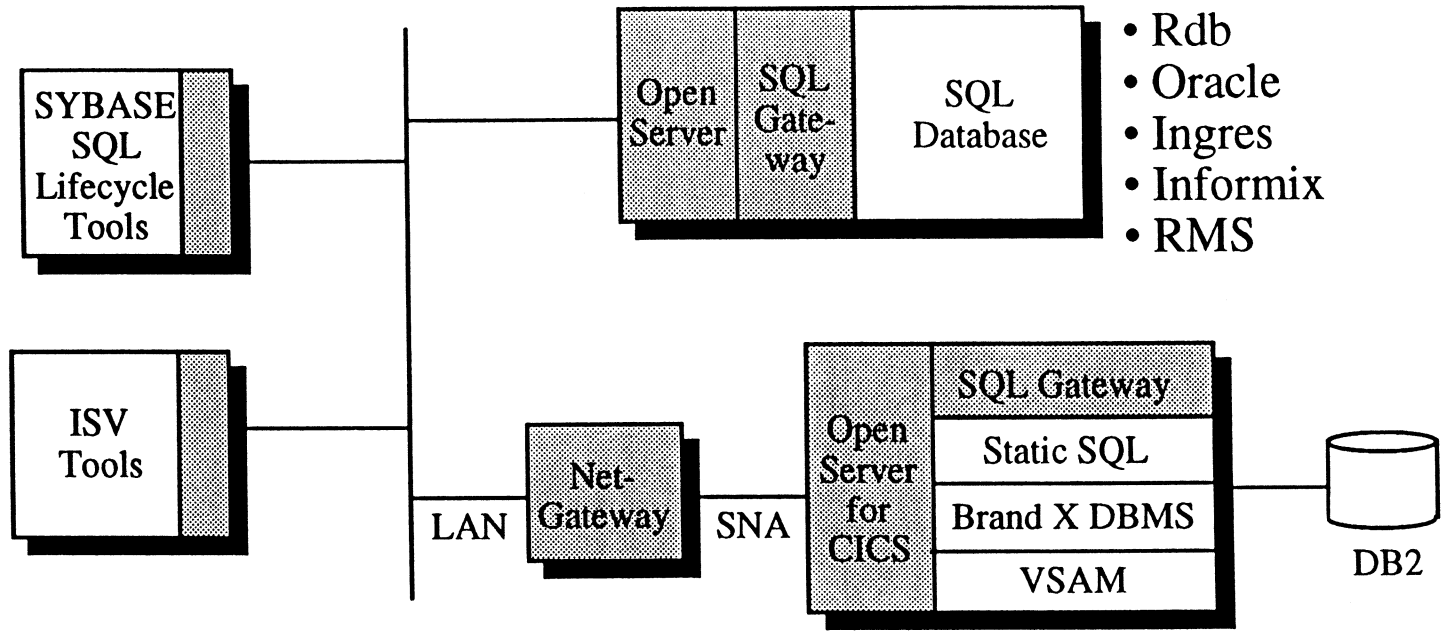
- transparent error recovery, connection services, and command data-stream formats and network protocols
- both SQL and Database RPCs within the same connection
- network and connection management transparency
- multiple programming interfaces, including Embedded SQL and Call-Level interfaces
- Open Client is supported by over 600 ISVs and VARs providing tools and services

SYBASE Open Server is a Server Application Toolkit. Open Server goes beyond just integrating SQL databases. Open Server integrates any data source or computer service into the on-line enterprise, while maintaining high performance and control. Open Server supports:

- SQL and Database RPCs
- High performance mult-threaded server development
- Shields application developers from network development
- Integrates realtime changes through Event Notification

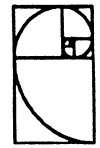
This architecture provides access to ANY data source or application, while giving users and programmers ease-of-use for higher productivity. With support for RPCs and SQL statements, programmers can maintain integrity and control while integrating SYBASE and nonSYBASE data.

Integrates New & Existing Technology



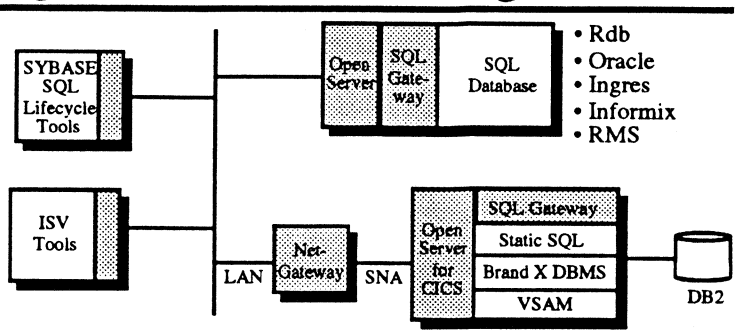
SYBASE Open Gateways Provide

- Fully Transparent SQL Access to Major RDBMSs
- High Performance Static SQL Access to DB2
- Common Format for:
  - Syntax, Data Types, Errors, Catalogues



MFCS.16

Integrates New & Existing Technology



SYBASE Open Gateways Provide

- Fully Transparent SQL Access to Major RDBMSs
- High Performance Static SQL Access to DB2
- Common Format for:
  - Syntax, Data Types, Errors, Catalogues

MFCS 16

Notes:

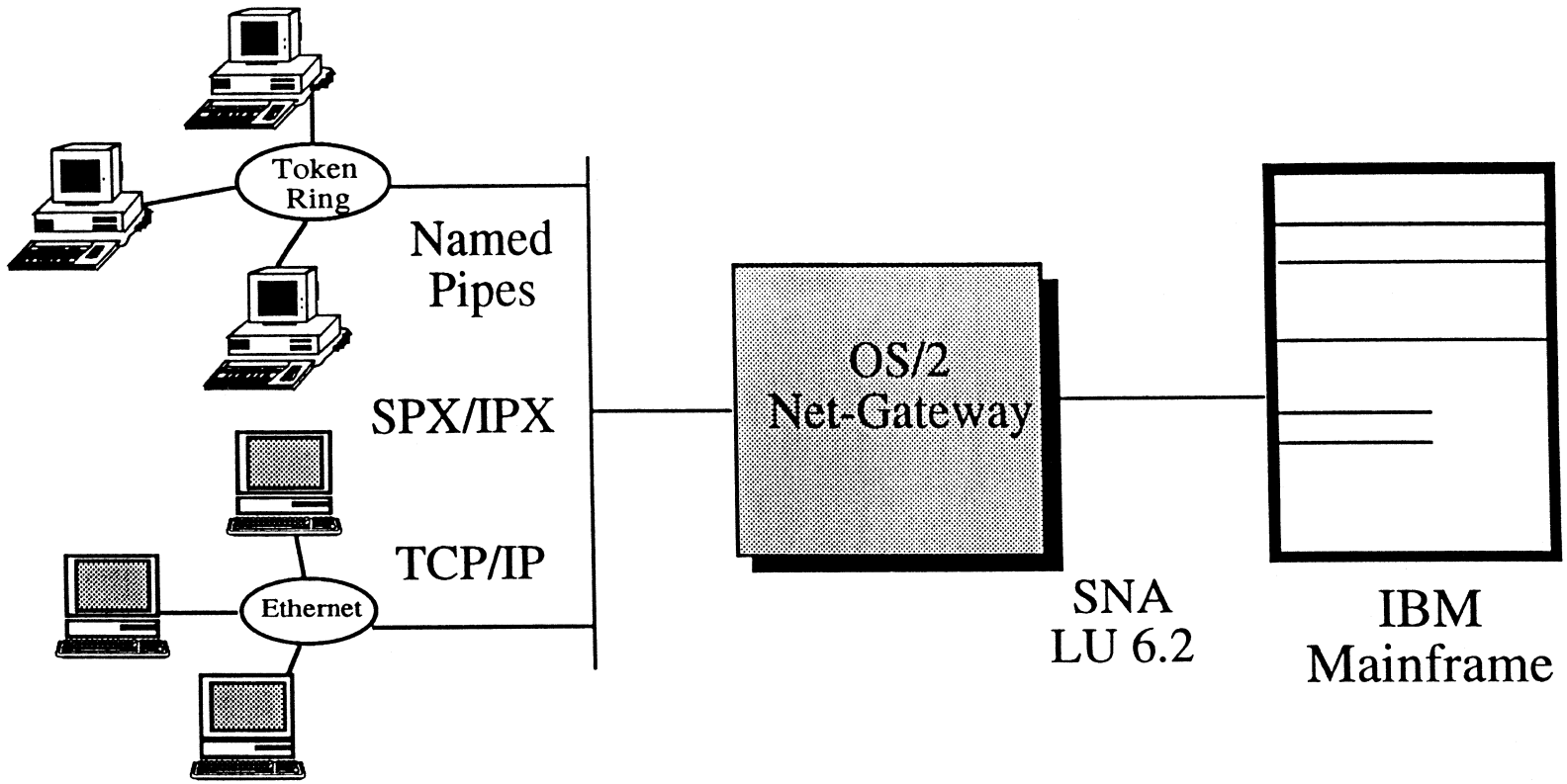
SYBASE Open Gateways are based on the SYBASE Open Server product. This enable Sybase to build gateways to key databases and integrate them into the SYBASE On-line Enterprise as if they were a SQL Server. This enables customers to access legacy data and new technology while maintaining a high level of security and control.

Sybase provides support for a number of databases including: Oracle, Ingres, Informix, RMS and RDB. Sybase provides static and dynamic access to DB2 data on the IBM mainframe and access to any other IBM application with the Open Server for CICS.

The products included in this architecture are:

- SYBASE Open Gateway to Oracle, RMS, RDB, Ingres and Informix
- SYBASE Open Server for CICS for high performance access to any CICS application (including static DB2 procedures)
- SYBASE Open Gateway for DB2 for fully transparent read and write access to DB2 data (includes Open Server for CICS and the SQL Gateway to DB2)
- SYBASE Net-Gateway which provides LAN to SNA protocol conversion and CICS security and administration for incoming requests

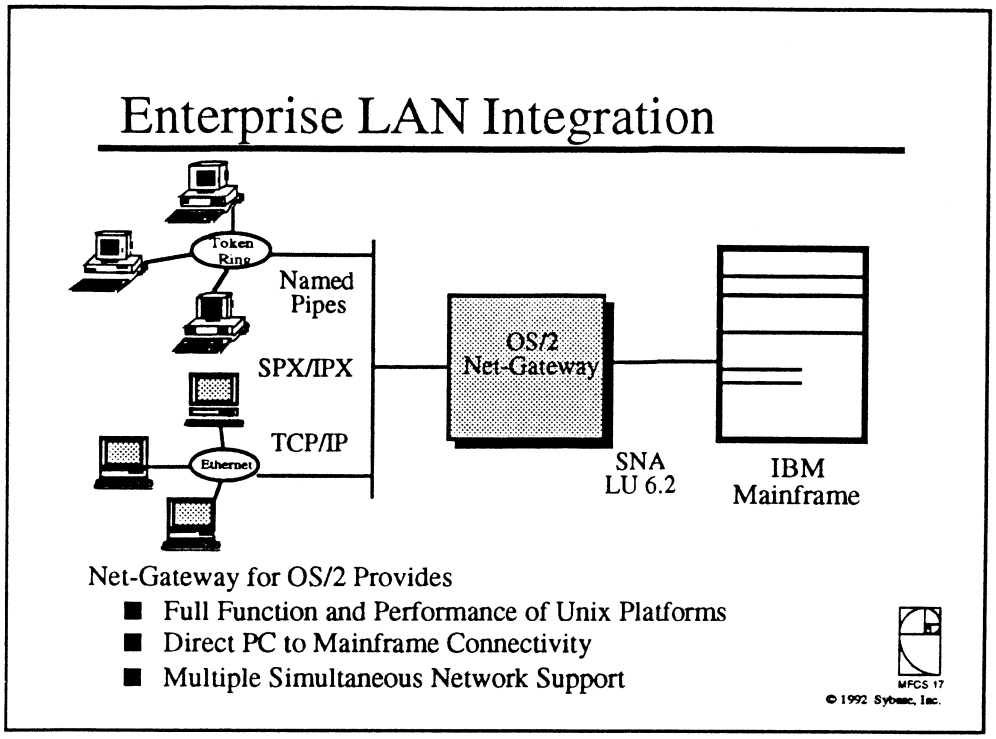
Enterprise LAN Integration



Net-Gateway for OS/2 Provides

- Full Function and Performance of Unix Platforms
- Direct PC to Mainframe Connectivity
- Multiple Simultaneous Network Support

MFCS.17



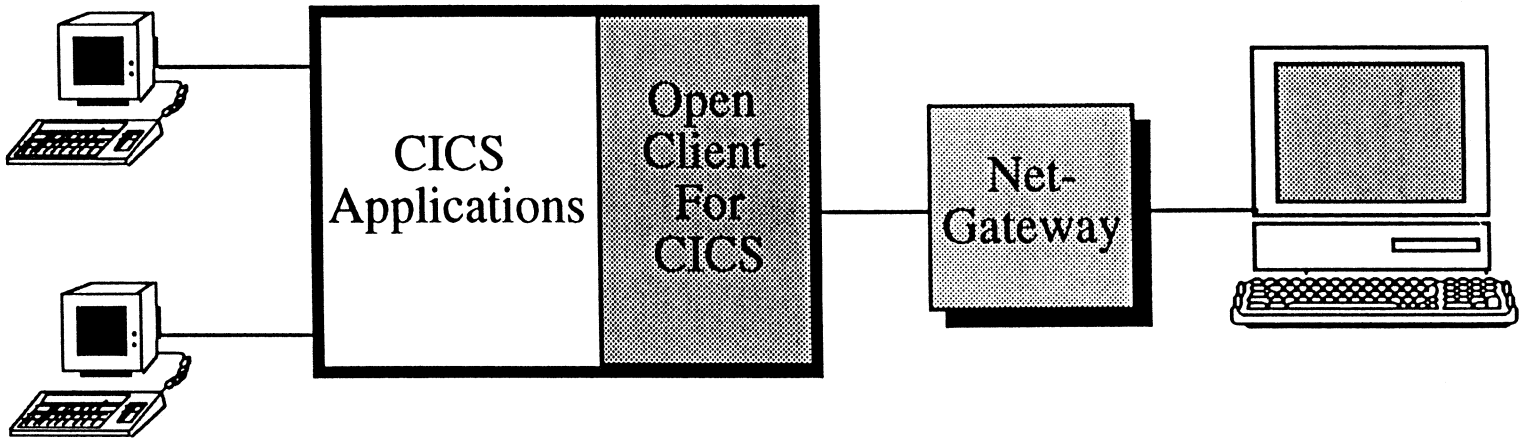
Key Points:

- Net-Gateway for OS/2 is a full 32bit OS/2 2.0
- Provides PC LAN networks with direct access to mainframe data and applications
- Supports multiple simultaneous networks from a single Net-Gateway

Notes:

The SYBASE **Net-Gateway** product is supported on the OS/2 platform. The Net-Gateway on OS/2 is a full 32-bit OS/2 2.0 application which provide the same level of performance and functionality of UNIX platforms. In addition, this product provides advanced networking support, including support for multiple networks and native PC protocols. PC protocols currently supported include TCP/IP, Novell SPX/IPX and Named Pipes (check your Product News for new protocols).

Full Mainframe Integration

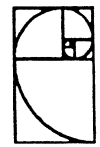


3270 Users
CICS Applications

Lan-based
Data and Applications

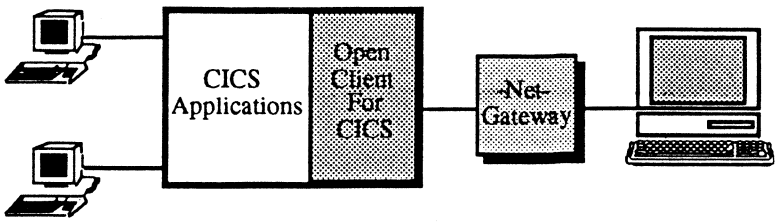
SYBASE Open Client for CICS Provides

- 3270 Terminal Access to Lan-based Data and Applications
- CICS Application Updates to Data on the LAN
- LAN and Mainframe Resource Synchronization



MFCS.18

Full Mainframe Integration



3270 Users
CICS Applications

Lan-based
Data and Applications

SYBASE Open Client for CICS Provides

- 3270 Terminal Access to Lan-based Data and Applications
- CICS Application Updates to Data on the LAN
- LAN and Mainframe Resource Synchronization

MFCS 18

Key Points:

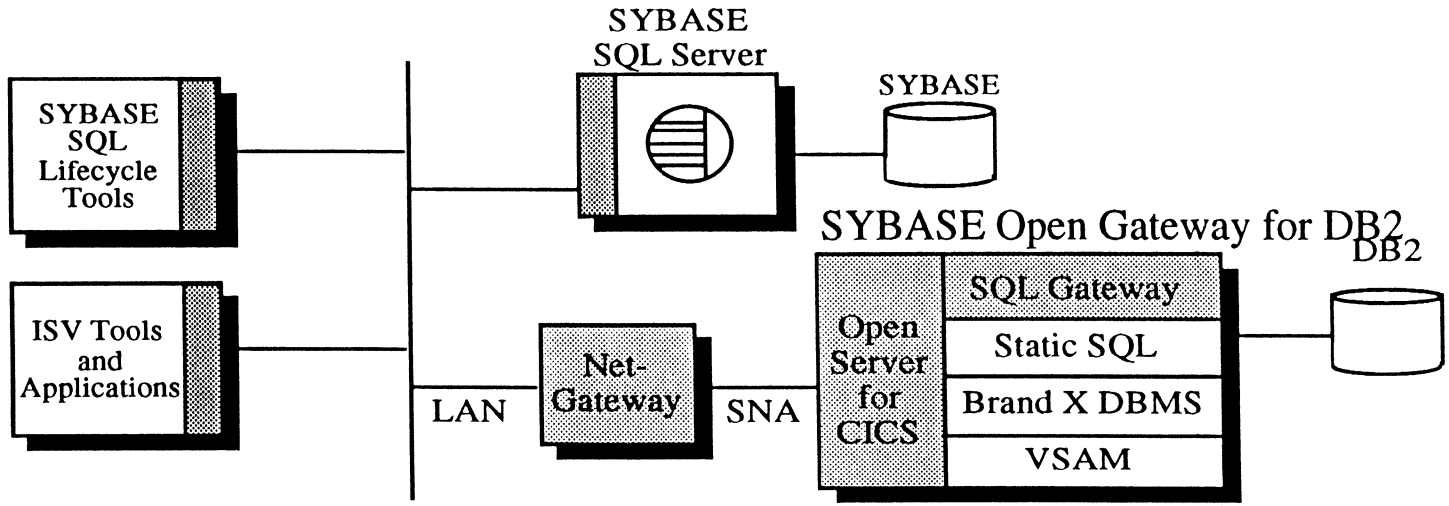
- enables the IBM mainframe to be a client in the SYBASE Client/Server architecture.
- lets 3270 terminal users access LAN data and services.
- helps organizations integrate legacy technology and emerging technology.

Notes:

One of the most exciting products Sybase has released is the **Open Client for CICS**. As with the Open Client on the LAN (UNIX, VMS, PCs, etc.), Open Client for CICS enables applications and users access to any SYBASE Server (SQL Server and ANY Open Server application). In the case of the IBM mainframe, some of those users may be using 3270 terminal. Now they can access SYBASE through interactive SQL or CICS applications.

With this architecture, mainframe users and programmers can also access new emerging technology such as x.500 services, new LAN EMAIL applications or new data sources.

Increases Resource Availability

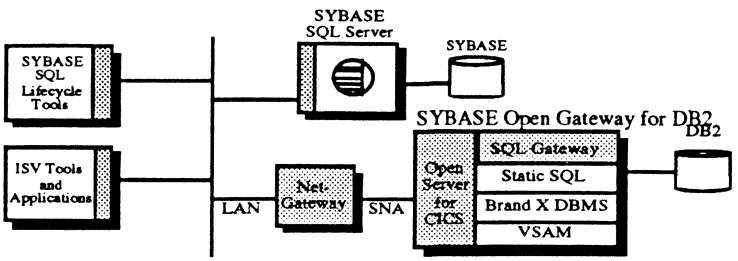


SYBASE Open Gateway for DB2 Provides

- Database Transparency via ODBC Catalog Support
- Support for National Languages
- Direct CICS to CICS Application Communication Through Open Client and Open Server for CICS

MFCS.19

Increases Resource Availability



SYBASE Open Gateway for DB2 Provides

- Database Transparency via ODBC Catalog Support
- Support for National Languages
- Direct CICS to CICS Application Communication Through Open Client and Open Server for CICS

MFCS 19

Key Points:

- Easier tools integration with a higher level of data transparency
- Sybase is taking the leading step in providing standard transparent access from any tool to SYBASE and mainframe data

Notes:

For over 2 years, Sybase has provided turnkey read/write access to DB2 through the Open Gateway for DB2. This year, Sybase enhanced this product to enable easier tools integration and higher availability. The Open Gateway for DB2 now provides support for:

- Microsoft's ODBC catalog stored procedures which provide data dictionary (or catalog) transparency between SYBASE and DB2. This means that tools vendors can easily write tools that integrate both SYBASE and DB2 because the structure of the data in DB2 looks like the structure of SYBASE data.

- Support for national (international) data and language commands

Another feature of this product is that programmers can use Open Client and Open Server on the same mainframe for CICS application-to-application communication. In other words, Sybase can be the mainframe application-to-application integrator as well as the LAN and mainframe application-to-application integrator.

Topics

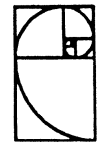
The Challenge of Enterprise Client/Server

The Right Architecture

SYBASE Advances the State of the Art

Enterprise Client/Server Successes

Summary



Topics

The Challenge of Enterprise Client/Server

The Right Architecture

SYBASE Advances the State of the Art

Enterprise Client/Server Successes

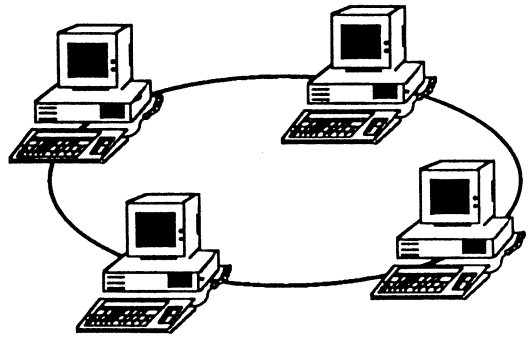
Summary

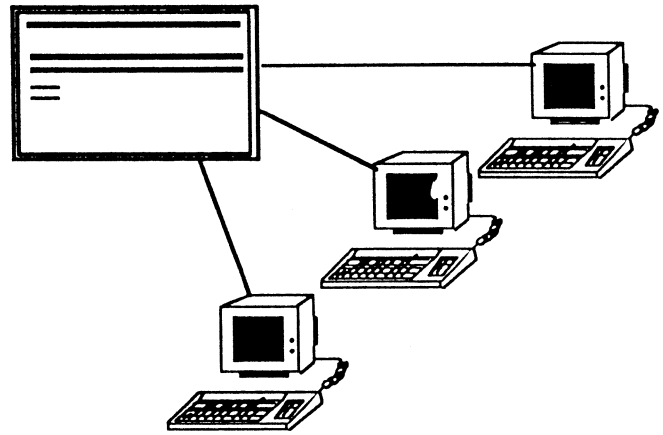
MFCS 20

Notes:

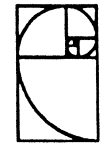
Now, let's look at some specific mainframe integration success stories

Corporate Integration Challenge



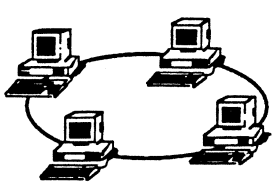


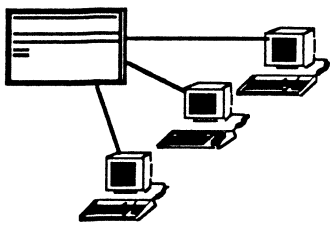
- Powerful Desktop Access into Corporate Information
  - DB2
  - IMS/DB
  - Workgroup Data And Applications
- Robust Performance
- Flexible, Transparent PC Tools integration
- Flexible LAN Protocol Support



MFCS.21

Corporate Integration Challenge





- Powerful Desktop Access into Corporate Information
  - DB2
  - IMS/DB
  - Workgroup Data And Applications
- Robust Performance
- Flexible, Transparent PC Tools integration
- Flexible LAN Protocol Support

MFCS 21

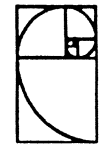
Key Points:

• Most Fortune 500 and 100 customers have more than one database and/or data source on multiple platforms.

• These customers need easy access to diverse data sources from PC applications while maintaining high performance and control.

SYBASE Integration Solution

	SYBASE Product	SYBASE Result
Access to Diverse Data	Open Server for CICS and Open Gateway to DB2	• High Performance Production Applications AND Decision Support
Access to Mainframe Data and Applications	Open Server for CICS, Open Client and SQL Server	• Evolutionary Migration • Fully Integrated Systems • High System Integrity
Transparent Tools Integration	ODBC catalog stored procedures and Certification Program	• More Tools Support • Application and Tools Portability
Flexible PC LAN Protocol Support	Net-Gateway for OS/2	• Easy Access to the Mainframe from the LAN • United Corporate Information



MFCS.22

SYBASE Integration Solution

	SYBASE Product	SYBASE Result
Access to Diverse Data	Open Server for CICS and Open Gateway to DB2	• High Performance Production Applications AND Decision Support
Access to Mainframe Data and Applications	Open Server for CICS, Open Client and SQL Server	• Evolutionary Migration • Fully Integrated Systems • High System Integrity
Transparent Tools Integration	ODBC catalog stored procedures and Certification Program	• More Tools Support • Application and Tools Portability
Flexible PC LAN Protocol Support	Net-Gateway for OS/2	• Easy Access to the Mainframe from the LAN • United Corporate Information

MFCS.22

Key Points:

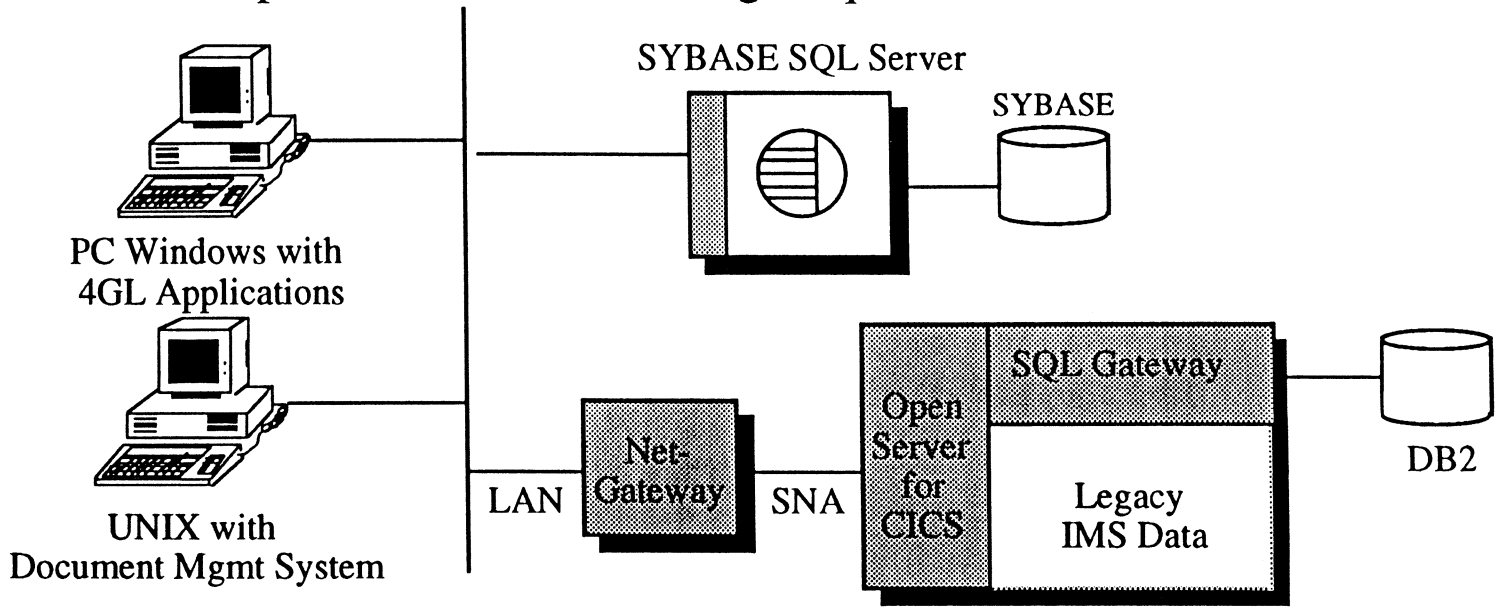
Sybase provides access to diverse data sources and applications through our Open Interoperability products

- Sybase is unique in providing access not only to mainframe data, but also to mainframe applications and other data sources through the Open Server for CICS product

- Sybase also provides easy access from LAN and PC tools through support for standard catalog stored procedures and support for native PC network access to the mainframe.

Fast, Reliable Response Example

Major International Petroleum Company
Need: To Respond to Federal Auditing Requirements to Minimize Fines



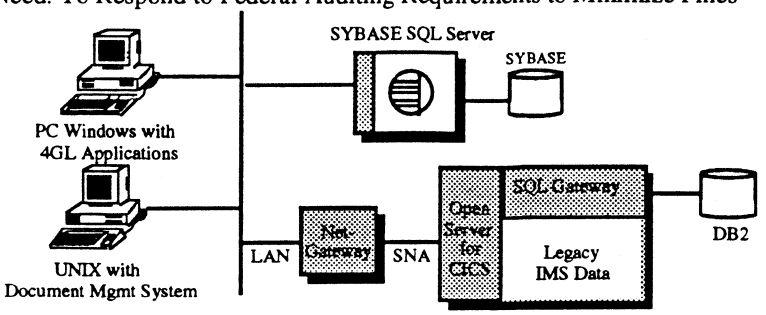
- Complete Integration of All Data and Applications
- Flexibility to Integrate new LAN Protocols
- Total System-wide Scalability

Result: Highly Responsive, Integrated Manufacturing Control System

MFCS.23

Fast, Reliable Response Example

Major International Petroleum Company
Need: To Respond to Federal Auditing Requirements to Minimize Fines



- Complete Integration of All Data and Applications
- Flexibility to Integrate new LAN Protocols
- Total System-wide Scalability

Result: Highly Responsive, Integrated Manufacturing Control System

Key Points:

- High Performance integration of LAN and mainframe data, including DB2 and IMS
- Access to SYBASE and mainframe from native PC applications and networks

Notes:

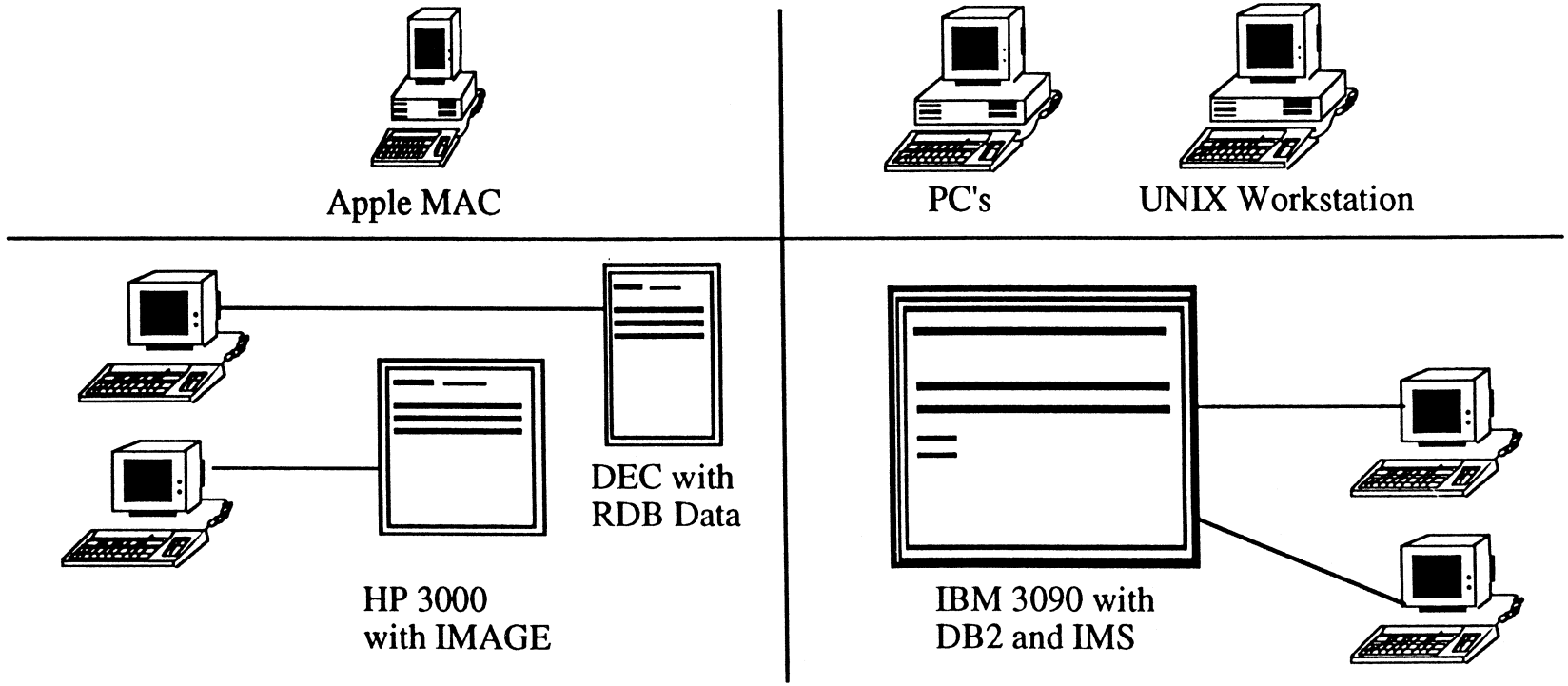
A major international petroleum company (who considers their use of SYBASE a competitive advantage) is using SYBASE to integrate data from the SQL Server on the LAN with mainframe data.

One of their challenges was the need to respond to government audits within the specified amount of time in order to avoid fines for returning a late audit.

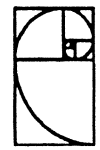
The basis of this system is to track the entire process (resources and personnel) associated with each refinery. Their major requirement was to be able to integrate diverse data from the LAN and the mainframe in order the respond to the federal government audits.

With Sybase they are able to accomplish complete integration while maintaining a high level of performance and control.

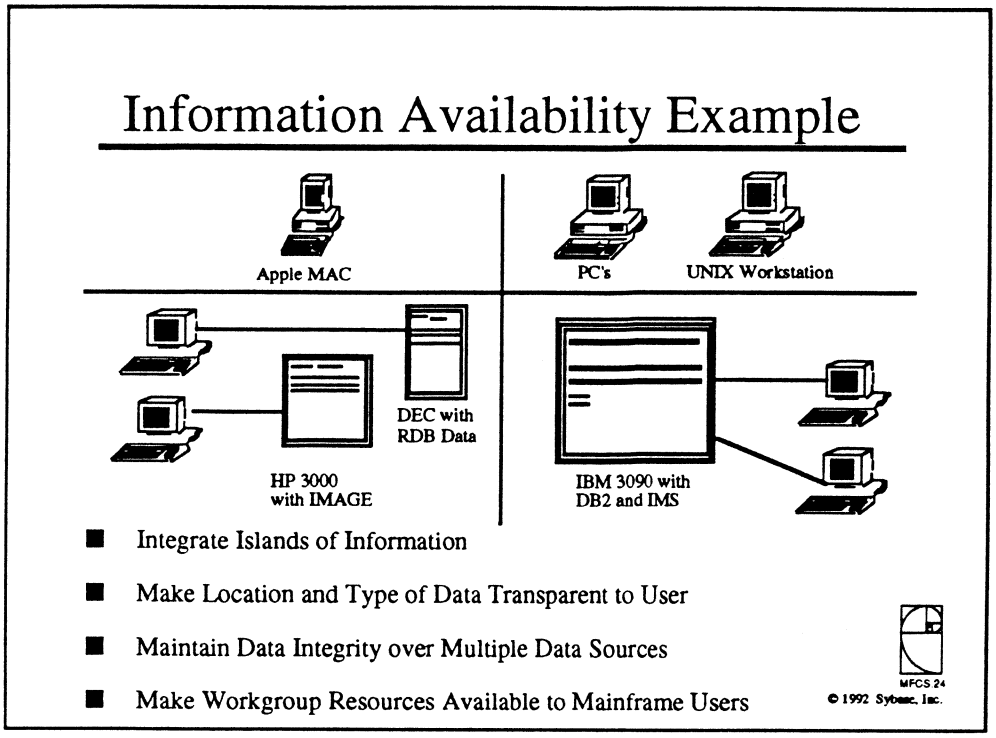
Information Availability Example



- Integrate Islands of Information
- Make Location and Type of Data Transparent to User
- Maintain Data Integrity over Multiple Data Sources
- Make Workgroup Resources Available to Mainframe Users



MFCS.24



Key Points:

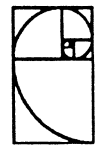
- Integrating the entire enterprise, including LAN to mainframe and mainframe to LAN
- Integrating diverse data with non data applications
- Maintaining high performance and control

Notes:

Another customer, Martin Marietta, needed to integrate diverse data (Turbo IMAGE, RDB) on a variety of platforms, including HP, DEC. They needed to integrate the islands of information throughout their enterprise while maintaining a high level of data availability and integrity.

SYBASE Availability Solution

	SYBASE	SYBASE Result
Read and Write Access to ANY Data Source	Open Server and the Open Gateways	• Complete Data Integration • More Competitive Solutions • Data Integrity and Security
Mainframe access to LAN Data or Services	Open Client for CICS	• Integration of Mainframe into LAN Applications • Access to LAN for 3270 Users
Integration of DBMS and Non-Data Services (X.500)	tightly integrated RPCs and SQL commands	• Total Application Flexibility • Leverages Investments • Assurance of Solution Growth
Toolkit for building specialized Servers	Open Server and Open Client	• Custom IS Solution • High Performance Specialized Processing



MFCS.25

SYBASE Availability Solution

	SYBASE	SYBASE Result
Read and Write Access to ANY Data Source	Open Server and the Open Gateways	• Complete Data Integration • More Competitive Solutions • Data Integrity and Security
Mainframe access to LAN Data or Services	Open Client for CICS	• Integration of Mainframe into LAN Applications • Access to LAN for 3270 Users
Integration of DBMS and Non-Data Services (X.500)	tightly integrated RPCs and SQL commands	• Total Application Flexibility • Leverages Investments • Assurance of Solution Growth
Toolkit for building specialized Servers	Open Server and Open Client	• Custom IS Solution • High Performance Specialized Processing

MFCS 25

Key Points:

- Sybase provides access to any data source and gateways to major databases. In addition, Sybase has cultivated a number of third party vendors who also write gateways to data sources based on the Open Server technology

- One of the unique features of SYBASE is that with the Open Client for CICS mainframe applications and users can access any SYBASE Servers (including SQL Server, Open Gateways and Open Server applications). This expands the mainframe and LAN interoperability to a new level.

- Sybase provides a way for users on the mainframe to access LAN data and services through RPCs and SQL statements.

- Sybase provides customers with a framework for accessing any data source or service. Sybase makes it easy to handle the complexity of developing network-based client/server applications.

OmniSQL Gateway 10.0

Product Overview

By Steve Olson

Overview - What is Omni?

- Omni is an Open Server application that attempts to present a common interface and appearance to multiple sources of data

  "A common interface from the desktop to the Enterprise"

- Omni enables applications written for SQL Server to access other SQL and non-SQL Server data sources

- Omni is not a data manager. It can be thought of more properly as a 'schema manager'.

- Engineering design goal is to appear identical to a SQL Server - we're almost there today.

Overview - Background

- Early versions of SQL Gateways enable Open Client applications to interact with non-Sybase databases.

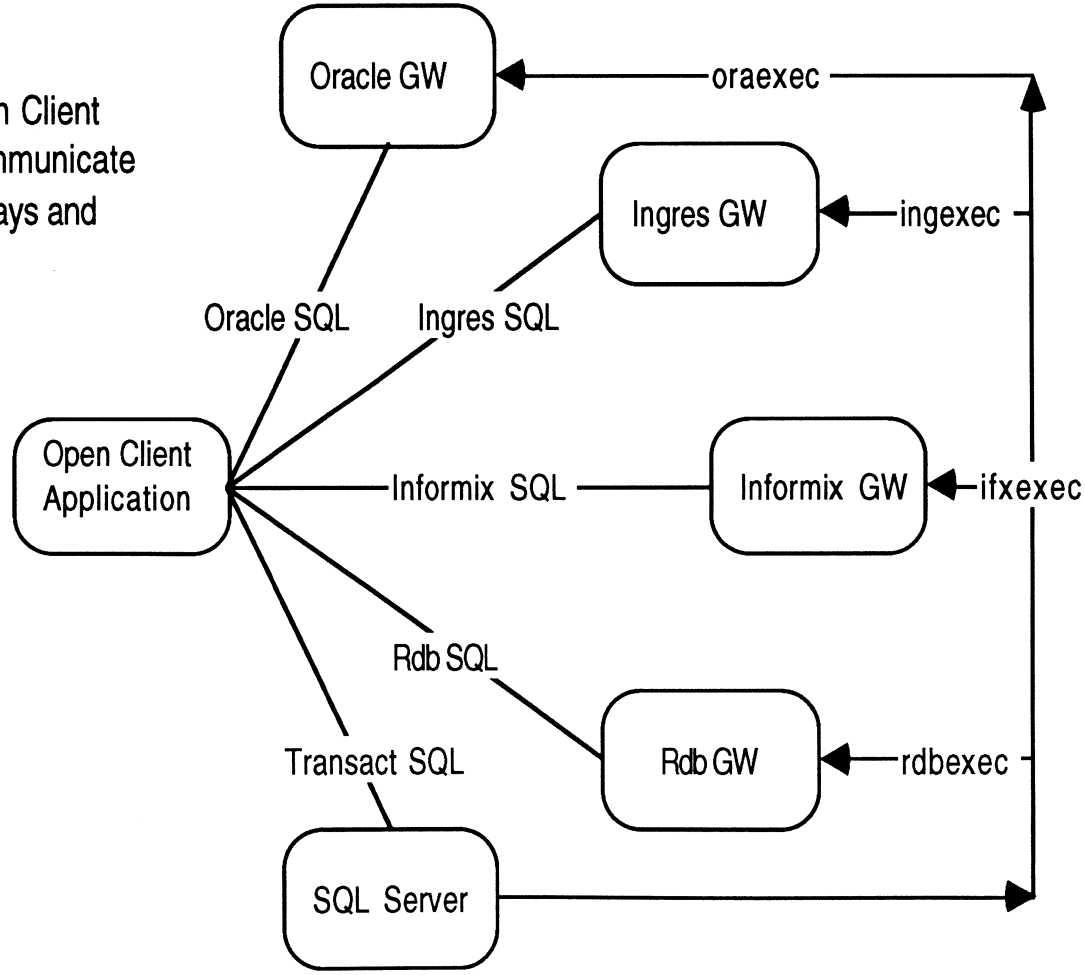
  Oracle
  Ingres
  Rdb
  Informix
  DB2

- Application are required to send native SQL to the gateway; no translation performed

- For each source of data, a separate Open Client connection must be established

Overview - Gateway Architecture

client uses Open Client interface to communicate with SQL Gateways and SQL Server



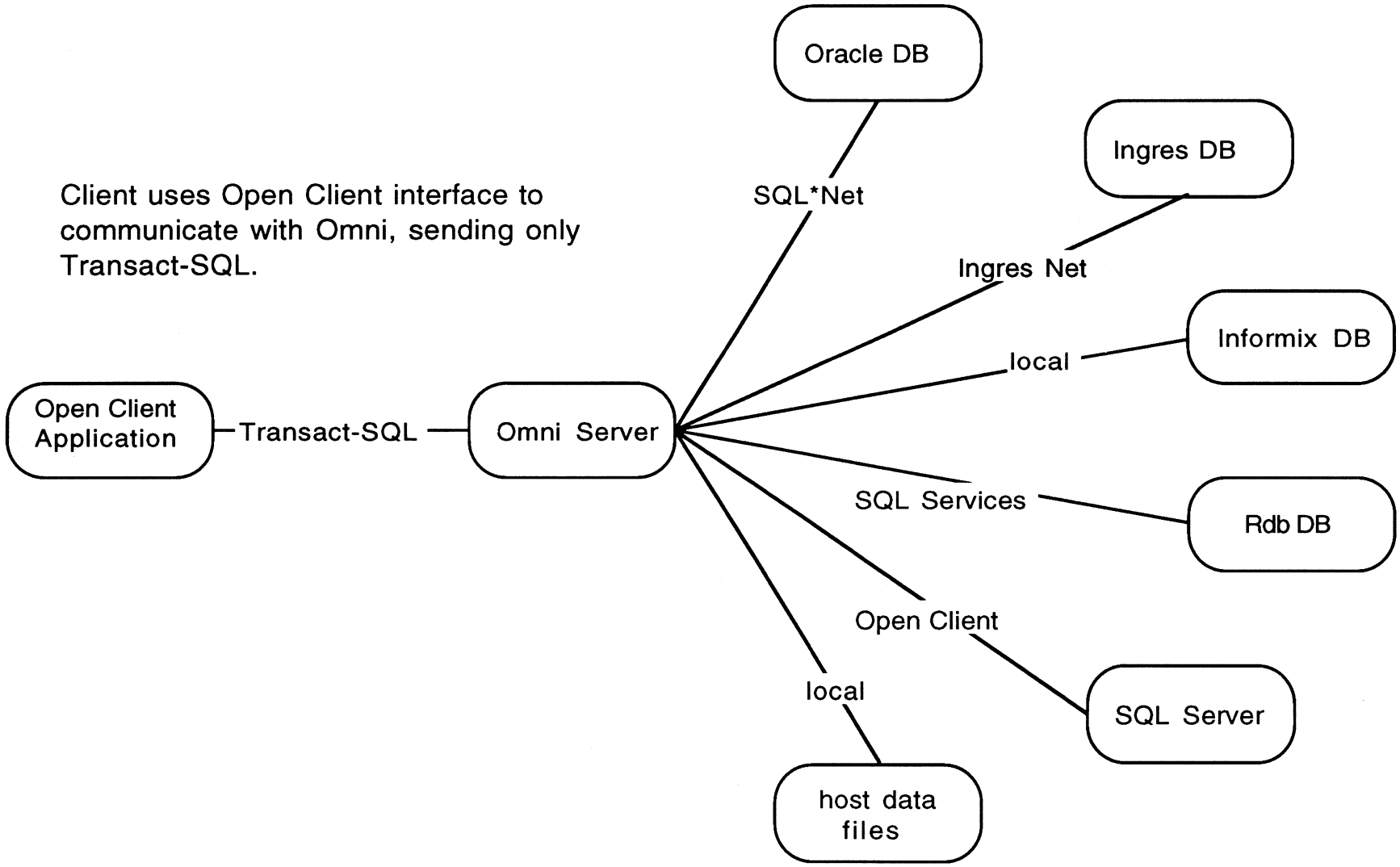
RPC's sent from SQL Server to SQL Gateways

Overview - OmniSQL Gateway

- Addresses the problem of non-uniform flavors of SQL when addressing various RDBMS through SQL gateways
- Single Language - Transact-SQL - only requirement for the application
- Access to remote heterogeneous data sources handled transparently by OmniSQL Gateway.
- To an Open Client application, the OmniSQL Gateway looks and behaves very much like a SQL Server.

Overview - Omni Architecture

Client uses Open Client interface to communicate with Omni, sending only Transact-SQL.

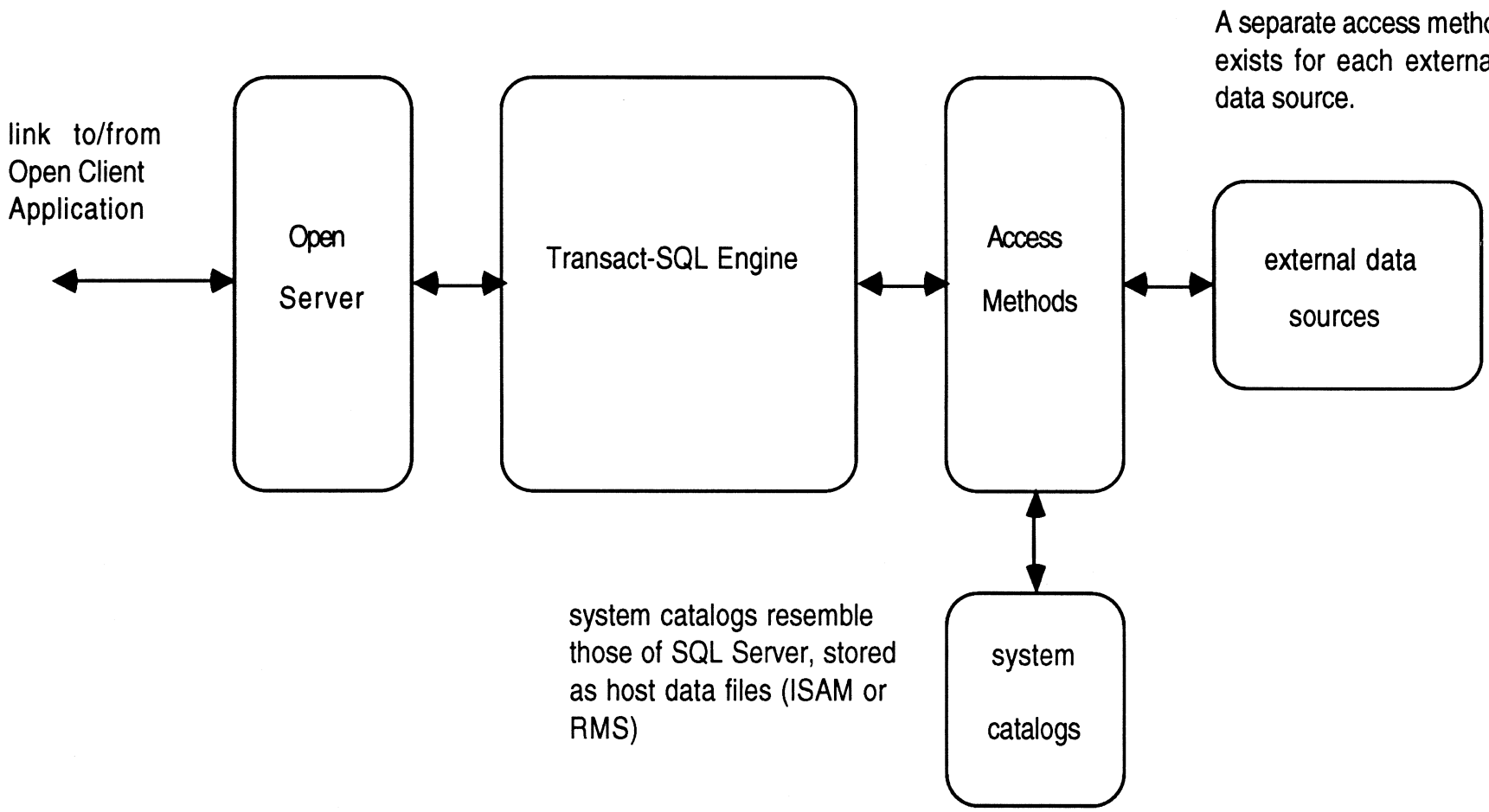


Overview - Omni Architecture

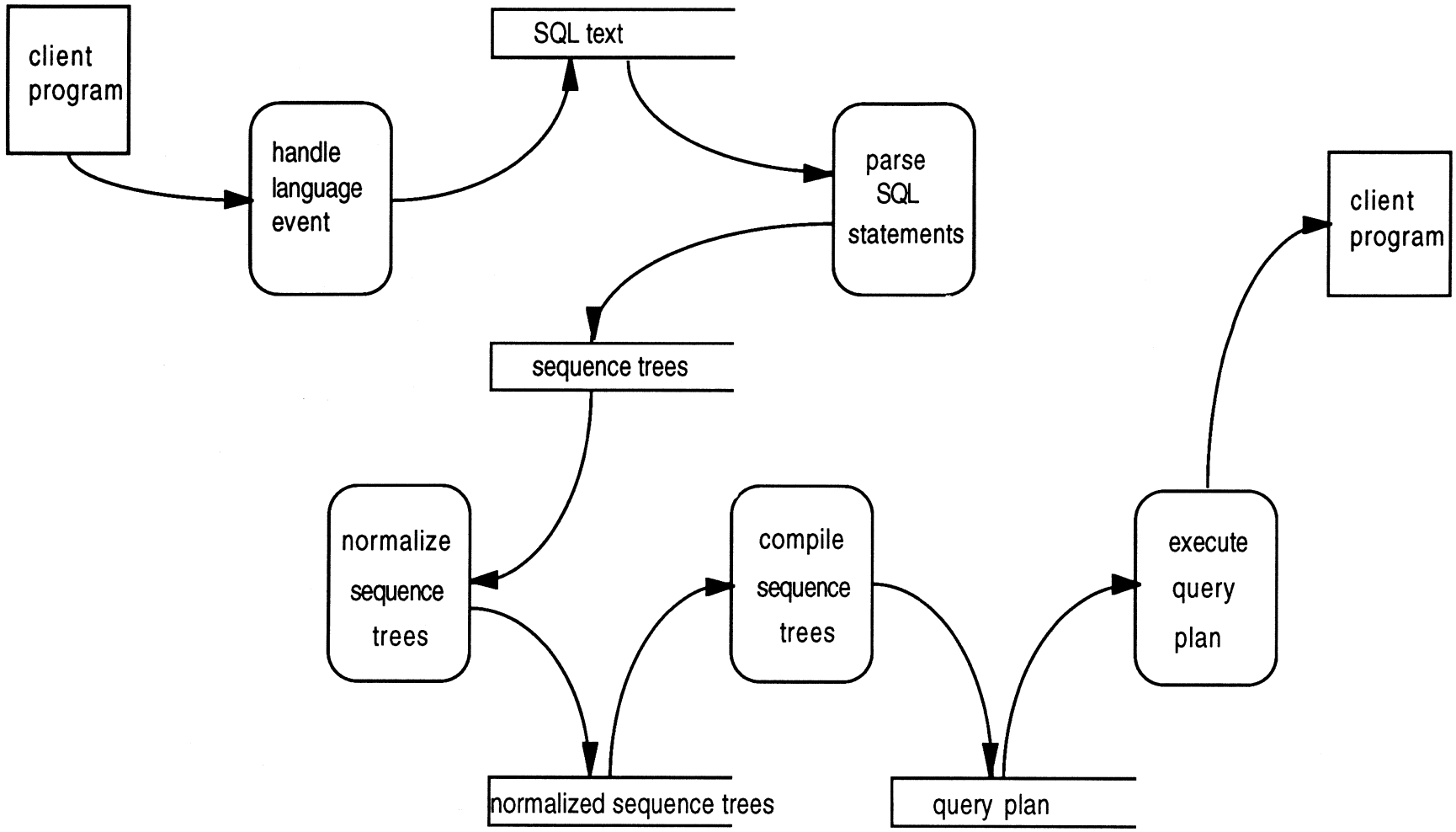
Consists of three primary components:

- Open Server 2.0.1, with EBF's
- Transact-SQL 'Engine' - derived from SQL Server version 4.8:
  - parser
  - sequencer - normalize, compile, execute
- Access Modules - driver for each external data source

Overview - Omni Components



Overview - Language Handling



Overview - Supported Transact-SQL

- Nearly all Transact-SQL commands are supported for all data sources, including:
  - stored procedures
  - views
  - rules
  - defaults
  - triggers (limited capability)
- Read/Write Access to all data sources is provided:
  - select
  - insert
  - delete
  - update

Overview - Supported Transact-SQL

- Transaction Control Statements Supported:
  - begin transaction
  - commit transaction
  - rollback transaction
  - save transaction
  - prepare transaction
- Some limitations exist on different data sources

Overview - Supported Transact-SQL

- Utility Commands Supported:
  - create table
  - create existing table
  - create index
  - drop table
  - drop index
  - alter table
  - truncate table
  - create database
  - drop database

Overview - Supported Transact-SQL

- Miscellaneous Transact-SQL Support:
  - built-in functions
  - local and global variables
  - flow control statements
  - waitfor
  - many others

Overview - Enhanced Transact-SQL

- Enhanced Transact-SQL Support:
  - create existing table
  - create database [on directory]
- New Transact-SQL Support:
  - connect to remote_server_name
  - disconnect

Overview - Unsupported SQL

- Text and image data types
- Compute rows
- Browse mode
- Bulk copy interface
- Two phase commit services
- dump/load database/transaction log
- ALTER DATABASE
- DISK INIT and relatives

Overview - System Catalogs

- Omni's system catalogs are almost equivalent to SQL Server:
  - all but two SQL Server's catalogs are present; most are supported, but some are present for compatibility only
  - two new catalogs are unique to Omni
- On Unix platforms the system catalogs are stored in ISAM files
- On VAX/VMS the system catalogs are stored in RMS files

Overview - Supported Catalogs

- System tables in all databases:
  - sysalternates
  - syscolumns
  - syscomments
  - sysdepends
  - sysindexes
  - syskeys
  - sysobjects
  - sysprocedures
  - sysprotects
  - systypes
  - sysusers

Overview - Supported Catalogs

- System tables in *master* database only:
  - sysconfigures
  - sysdatabases
  - syslanguages
  - syslogins
  - sysmessages
  - sysprocesses
  - sysremotelogins
  - sysservers

Overview - New Catalogs

- System tables in all databases:
  - systabledefs
- System tables in *master* database only:
  - sysexternlogins

Overview - Unsupported Catalogs

- The following system tables exist but are not used:
  - syssegments
  - syscharsets
  - sysdevices
  - syslocks
  - sysusages
- The following system tables do NOT exist:
  - syscurconfigs
  - syslogs

Overview - System Procedures

- Omni supports all SQL Server system procedures, except for the following:
  - sp_addsegment
  - sp_addumpdevice
  - sp_diskdefault
  - sp_dropdevice
  - sp_dropsegment
  - sp_extendsegment
  - sp_helpdevice
  - sp_helplog
  - sp_helpsegment
  - sp_lock
  - sp_logdevice
  - sp_placeobject
  - sp_spaceused

Overview - System Procedures

- The following system procedures are unique to Omni:
  - sp_addexternlogin
  - sp_addtabledef
  - sp_autoconnect
  - sp_dropexternlogin
  - sp_droptabledef
  - sp_fileinfo
  - sp_procio
  - sp_servertype
  - sp_tableio
  - sp_userio

Overview - Possible Uses

- Migration from one RDBMS to another:

```
sp_addtabledef target, "SYBASE.pubs.dbo.target"
go
select * into target from source_table
go
```

When transferring to SQL Server, the bulk copy interface is used internally

Overview - Possible Uses

- Coexistence with Multiple, Heterogeneous Systems:
  - location transparency
  - distributed joins
  - automatic data conversions
  - stored procedures used to encapsulate complex operations
  - rules and defaults are resolved within Omni and applied to all data sources

Overview - Administrative Steps

- Define remote server (data source)

```
sp_addserver 'ORACLE'

sp_servertype ORACLE, oracle6, native,
                "@T:maple:orasid"
```

- Define remote server login

```
sp_addexternlogin ORACLE, sa, system,
                        manager
```

- Should now be able to connect to server in passthru mode:

```
connect to ORACLE
```

Overview - Administrative Steps

- Define remote server tables to be accessed:

```
sp_addtabledef accounts,
                    "ORACLE..system.accounts"

create existing table accounts (
   name      varchar(30),
   id        int,
   balance   money,
   txn_date  datetime,
   etc, etc
)
```

- Grant access privileges to Omni table:

```
grant all on accounts to public
```

Omni Today

- Provided on four platforms:
  - Sun4 SunOS 4.1.2
  - HP9000/800 HP-UX 9.0
  - RS/6000 AIX 3.2
  - VAX/VMS 5.5
  - NCR 3000 SRV4
  - Novell NetWare
- Supports data from:
  - Sybase SQL Server 4.x and above
  - Oracle versions 6, 7
  - DB2 via New Gateway 2.0 and above
  - C-ISAM files (Unix)
  - RMS files (VMS)
  - Ingres
  - Generic access modules

Omni Tomorrow

- Release 10.1 planned for Q2 1994
  - Full System 10 compatibility
    - SQL Server 10.0 code line
    - Uses Open Client/Server 10.0
  - Expanded platform coverage (all core platforms)
  - Full SQL Server datatype support
  - Computed result sets
  - On-going performance enhancements
  - Full internationalization support
  - Extended non-Sybase database coverage via Open Server-based access modules

Omni Tomorrow, continued

- OmniSQL Toolkit
  - Includes full Transact-SQL parser library
  - De-coupled architecture
  - Allows user customizable data access
  - Common module interface to facilitate interchangeable modules
  - Used to develop Omni-compatible Access Modules
  - Consistent Sybase Client-Server Design

- Standard disclaimer:
  - feature set(s) of future releases has not been finalized

Installation - Overview

- Regardless of platform the basic steps are as follows:

  - verify network
  - add sybase user account
  - add SYBASE to environment
  - load Omni software from tape
  - run *omnibuild*
  - create/modify interfaces file
  - run *omniserv*
  - load installmaster
  - load installmodel
  - load installpubs
  - load inscsprocs - ODBC catalog stored procedures

Omni Configuration

- References:

  "OmniSQL Gateway Release Reference Manual for Unix Platforms"

  "OmniSQL Gateway Release Reference Manual for VAX/VMS"

  "System Administration Guide for SYBASE OmniSQL Gateway"

Configuration Overview

- sp_configure configuration variables
- Adding and Configuring remote servers
- Configuring logins **TO** remote servers
- Configuring logins **FROM** remote servers

Configuration - sp_configure

- The use of *sp_configure* is exactly the same as SQL Server, the specific configuration variables differ.

- Omni configuration variables:

allow updates
lock timeout interval (VAX/VMS only)
open databases
open objects
procedure cache size
query buffer size
read regardless (VAX/VMS only)
remote access
remote connections
remote read buffers
remote sites
timer wakeup interval
user connections

Configuration - sp_configure

- Omni configuration variables which operate the **same** as SQL Server:

  allow updates
  open databases
  open objects
  remote access
  remote connections
  remote sites
  user connections

- Omni config variables that **differ** from SQL Server:

  lock timeout interval (VAX/VMS only)
  procedure cache size
  query buffer size
  read regardless (VAX/VMS only)
  remote read buffers
  timer wakeup interval

Configuration - sp_configure

- *lock timeout interval (VAX/VMS only)* configuration variable:

  Omni uses multi-streaming to enable multiple users to access the same files. This variable specifies the number of seconds any stream may wait for a locked RMS record. Zero indicates no waiting.

  Dynamic

  Default = 10 (seconds)

Configuration - sp_configure

- *read regardless (VAX/VMS only)* configuration variable:

  Allows Omni to retrieve RMS records that have been locked exclusively by another process. Enabling this feature will disable *lock timeout interval*, since no waiting for locked records will be done by RMS.

  Dynamic

  Default = 0 (disabled)

Configuration - sp_configure

- *procedure cache size* configuration variable:

  Sets the size of memory, in 2K pages, that Omni allocates from the operating system for use as a cache for stored procedures and as work space when compiling incoming Transact-SQL.

  Default = 500

- *query buffer size* configuration variable:

  Sets the size of memory, in 1K increments, that Omni allocates from the operating system for use as a cache for constructing SQL statements to issue to remote servers.

  Default = 4

Configuration - sp_configure

- *remote read buffers* configuration variable:

  Sets the number of packets which will be preread from remote connections. Default is adequate for virtually all cases.

  Default = 3

- *timer wakeup interval* configuration variable:

  Omni has a background timer which responds to the value of this variable. Primarily affects the effective granularity of the *waitfor* command.

  Default = 5

Configuration - Remote Servers

- In order to access a remote server it must first be defined. This is a multi-step process which consists of *sp_addserver* and *sp_servertype*. Optionally, *sp_serveroption* may be utilized.

- *sp_addserver* functions exactly as the SQL Server. It inserts a row into the sysservers table.

- *sp_servertype* is a system procedure unique to Omni. It is used to define a remote server's class and network access information.

- The valid server classes are:

sql_server	oracle6
oracle7	db2
ingres6	generic

Configuration - sp_servertype

```
sp_servertype server_name, class, interface,
    network_info
```

- server_name - name of server, as found in sysservers, for which a class is to be defined or modified
- class - class of the server. Legal values are *sql_server*, *oracle6, oracle7, ingres6, db2* and *generic*.
- interface - interface type to be used to access the remote server.

  oclient - indicates that open client is going to be used to access the remote server. Valid for *sql_server, db2* and *generic* classes.

  native - indicates that a remote server's native access should be used to access the remote server. Valid for *oracle6, oracle7* and *ingres6*.

Configuration - sp_servertype

```
sp_servertype server_name, class, interface,
    network_info
```

- network_info - varies depending on server class:

sql_server	the name of the remote server as found in the interfaces file.
oracle6/7	SQL*Net address of the Oracle database. Use " " if SQL*Net is not used
db2	Net-Gateway server name, as found in the interfaces file
ingres6	Name of directory that will be used to set the II_SYSTEM environment variable
generic	Generic Access Module, as found in interfaces file

Configuration - sp_servertype Example

```
sp_addserver OMNI, local
go

sp_addserver SYBASE
go

sp_servertype SYBASE, sql_server, oclient, SYB_PUBLIC
go

use master
go

checkpoint
go
```

Configuration - sp_serveroption

- Omni provides two options for remote servers:

  timeouts - works the same as SQL Server.

  writable - unique to Omni. Must be TRUE in order to perform write operations (create, insert, update, delete) on a remote server.

  Allows a remote server to be defined as 'READ ONLY' (if set FALSE).

  Defaults to TRUE for *sql_server* and *oracle6* classes.

  Defaults to FALSE for *db2* class.

Configuration - helpserver & dropserver

- Omni also provides both *sp_helpserver* and *sp_dropserver*. Each of these procedures operates the same as on a SQL Server.

Configuration - sp_addexternlogin

- *sp_addexternlogin* is used to instruct Omni how to log a given Omni user into a remote server.
- Inserts or updates a row in *sysexternlogins*

```
sp_addexternlogin server, loginname, externname,
    externpasswd
```

- *server* - name of remote server as found in sysservers
- *loginname* - name of Omni user as found in syslogins
- *externname* - login name to use with remote server
- *externpasswd* - login password to use with remote server

Configuration - sp_addremotelogin

- *sp_addremotelogin* is used to instruct Omni which remote SQL Server users can access the server, and what identity they will assume.
- *sp_addremotelogin* works the same as SQL Server.

Server Classes - Overview

- Server class *local*
- Server class *sql_server*
- Server class *oracel6, oracle7*
- Server class *db2*
- Server class *generic*
- Passthrough mode
- The *defgen* utility

Server Class - local

- The local server is the currently running Omni server
- Objects managed by the local server include host data files
  - on Unix, these are ISAM files
  - on VMS, these are RMS files
- Unless table mapping has been specified, it is assumed that the table about to be created is a *local* object
- Default mapping for local tables can be overridden using the *sp_addtabledef* procedure:

Server Class - local

```
sp_addtabledef table_name, "pathname"
```

- pathname can be any legal file specification:
  - read access is required by the Omni process
- If an existing object is an ISAM file, don't use the file's suffix in the pathname (.idx or .dat)
- During the *create existing* function, file attributes are obtained from the file system and used to update Omni's system catalogs.

Server Class - local - Unix systems

- Datatype mapping for *create table* ; Omni type (left) maps to type used by C-ISAM (right):

binary(n)	chartype(n)
bit	chartype(1)
char(n)	chartype(n)
datetime	2 longtype fields
float	doubletype
int	longtype
money	longtype, chartype(4)
real	floattype
smalldatetime	chartype(4)
smallint	inttype
smallmoney	longtype
timestamp	longtype
tinyint	chartype(1)
varbinary(n)	chartype(n)
varchar(n)	varchar(n)

Server Class - local - Unix systems

- Datatype mapping for *create existing table* ; ISAM type (left) maps to type used by Omni (right):

chartype(n)	binary(n)
chartype(n)	varbinary(n)
chartype(n)	char(n)
chartype(n)	varchar(n)
chartype(1)	tinyint
decimal	binary(n) (application must convert)
doubletype	float
floattype	real
inttype	smallint
longtype	int

- Type checking is not performed, you must know the layout of the ISAM records

Server Class - local - VMS systems

- Datatype mapping for *create table* ; Omni type (left) maps to type used by RMS (right):

binary(n)	string(n)
bit	string(1)
char(n)	string(n)
datetime	VAX date (quadword)
float	D-float
int	longword
money	quadword
real	float
smalldatetime	longword
smallint	word
smallmoney	word
timestamp	longword
tinyint	string(1)
varbinary(n)	string(n)
varchar(n)	string(n)

Server Class - local - VMS systems

- Datatype mapping for *create existing table* ; RMS type (left) maps to type used by Omni (right):

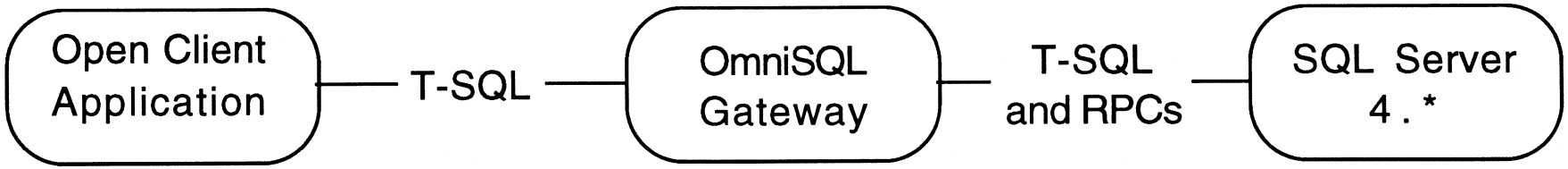
string(n)	binary(n)
string(n)	varbinary(n)
string(n)	char(n)
string(n)	varchar(n)
string(1)	tinyint
packed decimal	binary(n) (application must convert)
D-float	float
G-float	binary(8) (application must convert)
VAX date	datetime
word	smallint
longword	int
quadword (VAX date)	datetime

- Type checking is not performed, you must know the layout of the RMS records

- Column offsets and length are checked for match when key descriptor is found

Server Class - sql_server

Client uses Open Client interface to communicate with Omni, sending only Transact-SQL.



OmniSQL uses Open Client interface to communicate with SQL Servers, sending only T-SQL and RPCs.

Server Class - sql_server - data mapping

- Datatype mapping for *create table* and *alter table*; Omni type (left) maps to type used by SQL Server (right):

binary	binary
bit	bit
char	char
datetime	datetime
float	float
int	int
money	money
real	real
smalldatetime	smalldatetime
smallint	smallint
smallmoney	smallmoney
timestamp	timestamp
tinyint	tinyint
varbinary	varbinary
varchar	varchar

Server Class - sql_server - data mapping

- Datatype mapping for *create existing table*; SQL Server type (left) maps to type allowed by Omni (right):

binary	binary
bit	bit
char	char
datetime	datetime
float	float
image	UNSUPPORTED
int	int
money	money
real	real
smalldatetime	smalldatetime
smallint	smallint
smallmoney	smallmoney
text	UNSUPPORTED
timestamp	timestamp
tinyint	tinyint
varbinary	varbinary
varchar	varchar

- System 10 data types not yet supported

Server Class - sql_server - server definition

- Server definition via *sp_servertype*:

  server_name - local Omni server name

  class - always *sql_server* for Sybase servers

  interface - always *oclient*

  network_info - name of server as found in *interfaces* file

- No separate class yet for System 10 SQL Server

Server Class - sql_server - table mapping

- Mapping done via *sp_addtabledef*:

  table_name - local Omni table name

  remote_name - path to remote table or view in the form:

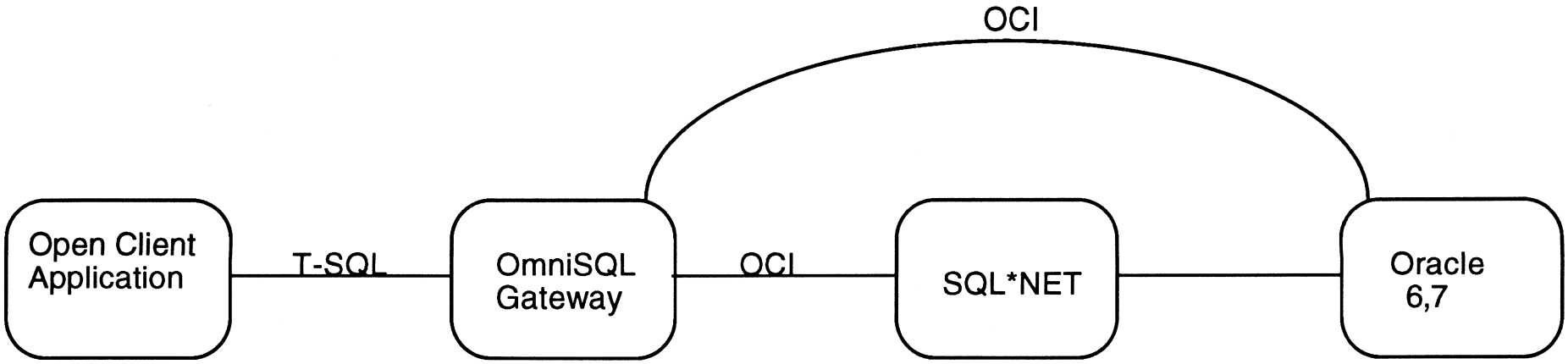
  server.database.owner.object

  where:

  server - name of Omni server
  database - name of database in remote SQL Server
  owner - name of owner of remote object
  object - name of either table or view in server

Server Class - oracle6, 7

Client uses Open Client interface to communicate with Omni, sending only Transact-SQL



OmniSQL uses Oracle's OCI interface to communicate with Oracle, either directly or via SQL*NET

Server Class - oracle6, 7 - data mapping

- Datatype mapping for *create table* or *alter table*; supplied Omni type (left) maps to type used by Oracle (right):

binary	raw
bit	number(1)
char	char
datetime	date
float	float
int	number(10)
money	number(19,4)
real	float
smalldatetime	date
smallint	number(5)
smallmoney	number(19,4)
tinyint	number(3)
varbinary	raw
varchar	char

Server Class - oracle6, 7 - data mapping

- Datatype mapping for *create existing table*; Oracle type (left) maps to type allowed by Omni (right):

char	char, varchar
rowid	char, varchar
number	bit
number	tinyint, smallint, int
number	real, float
number	smallmoney, money
date	smalldatetime, datetime
long	varchar(255)
raw	binary
longraw	varbinary(255)
varchar	varchar(255)

Server Class- oracle6,7- data mapping, passthru

- Datatype mapping for *passthrough mode*; Oracle type (left) maps to Omni type (right):

char	varchar
rowid	varchar
number	float
date	datetime
long	varchar(255); truncated if length > 255
raw	binary
longraw	varbinary(255); truncated if length > 255
varchar(oracle7)	varchar(255); truncated if length > 255

Server Class - oracle6, 7 - server definition

- Server definition via *sp_servertype*:

  server_name - local Omni server name

  class - either *oracle6 or oracle7* for Oracle servers

  interface - always *native*

  network_info - SQL*Net address of Oracle database, or ""

Server Class - oracle6, 7 - SQL*Net definition

- Network information provided by *sp_servertype* can specify SQL*Net connect string for local or remote Oracle servers

- If Oracle and Omni are on the same system, SQL*Net is not required. Use empty string ("") and set Oracle environment variables ORA_SID and ORA_SYSTEM before starting Omni

- SQL*Net string can specify either TCP/IP or DECNet protocols to be used:

  - TCP/IP:
  @T:host_name:ora_sid[,retry_cnt,buf_size,keepalive]

  - DECNet:
  @D:host_name-ORDNora_sid OR

  @D:host"name passwd"::"TASK=filename"

Server Class - oracle6, 7 - table mapping

- Mapping done via *sp_addtabledef*:

  table_name - local Omni table name

  remote_name - path to remote table or view in the form:

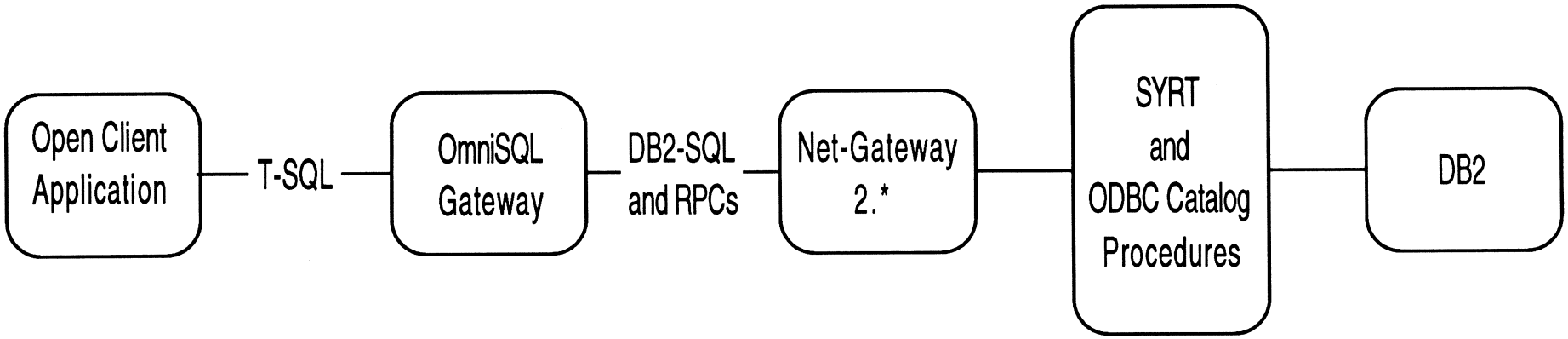
  server.database.owner.object

  where:

  server - name of Omni server
  database - ignored, since no Oracle support for databases
  owner - name of owner of remote object
  object - name of either table or view in Oracle

Server Class - db2

Client uses Open Client interface to communicate with Omni, sending only Transact-SQL.



OmniSQL uses Open Client interface to communicate with Net-Gateway, sending only DB2 and RPCs.

Server Class - db2 - data mapping

- Datatype mapping for *create table* and *alter table*; supplied Omni type (left) maps to type used by DB2 (right):

binary(n)	varchar(n*2); n <= 127
bit	char(1)
char(n)	char(n); n <= 254
datetime	timestamp
float	float
int	int
money	float
real	real
smalldatetime	timestamp
smallint	smallint
smallmoney	float
tinyint	smallint
varbinary(n)	varchar(n*2); n <= 127
varchar(n)	varchar(n); n <= 254

Server Class - db2 - data mapping

- Datatype mapping for *create existing table*; DB2 type (left) maps to type allowed by Omni (right):

int	int
smallint	smallint
float(n); n <= 21	real, float, money
float(n); n > 21	float, money
float	float, money
double precision	float, money
real	real, float, money
decimal; scale > 0	float, money
decimal; scale = 0	int, float, money
numeric; scale > 0	float, money
numeric; scale = 0	int, float, money
char	char, varchar
varchar	char, varchar
long varchar	char(255), varchar(255)

Server Class - db2 - data mapping

- Datatype mapping for *create existing table,* continued: DB2 type (left) maps to type allowed by Omni (right):

date	char, varchar, datetime (time=12:00am)
time	char, varchar, datetime (date=1/1/1900)
timestamp	char, varchar, datetime
tinyint	tinyint
graphic	UNSUPPORTED
vargraphic	UNSUPPORTED
long vargraphic	UNSUPPORTED

Server Class - db2 - server definition

- Server definition via *sp_servertype*:

  server_name - local Omni server name

  class - always *sql_server* for Sybase servers

  interface - always *oclient*

  network_info - name of Sybase Net Gateway as found in *interfaces* file

- Net Gateway version 2.0 is required

Server Class - db2 - table mapping

- Mapping done via *sp_addtabledef*:

  table_name - local Omni table name

  remote_name - path to remote table or view in the form:

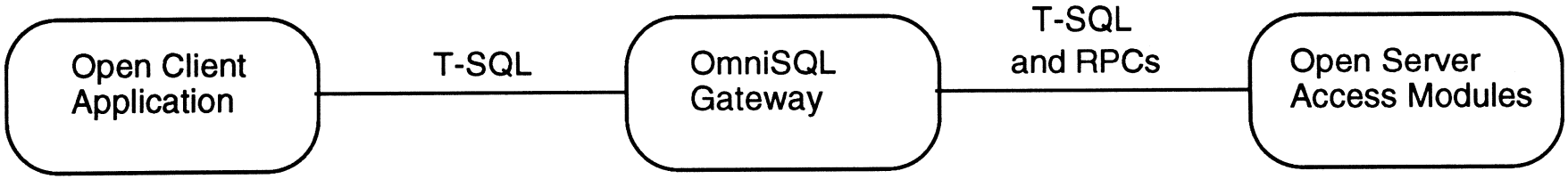
  server.database.owner.object[;aux1,aux2]

  where:

  server - name of Omni server
  database - location-name portion of DB2 table name
  owner - DB2 authorization id
  object - name of either table or view in server
  aux1 - DB2 database in which locate table
  aux2 - DB2 tablespace in which to locate table

Server Class - generic

Client uses Open Client interface to communicate with Omni, sending only Transact-SQL



OmniSQL uses Open Client interface to communicate with Open Server modules, sending only T-SQL and RPCs

Server Class - generic - data mapping

- Datatype mapping for *create table* and *alter table*; Omni type (left) maps to type used by generic Server (right):

binary	binary
bit	bit
char	char
datetime	datetime
float	float
int	int
money	money
real	real
smalldatetime	smalldatetime
smallint	smallint
smallmoney	smallmoney
timestamp	timestamp
tinyint	tinyint
varbinary	varbinary
varchar	varchar

Server Class - generic - data mapping

- Datatype mapping for *create existing table*; generic Server type (left) maps to type allowed by Omni (right):

binary	binary
bit	bit
char	char
datetime	datetime
float	float
image	UNSUPPORTED
int	int
money	money
real	real
smalldatetime	smalldatetime
smallint	smallint
smallmoney	smallmoney
text	UNSUPPORTED
timestamp	timestamp
tinyint	tinyint
varbinary	varbinary
varchar	varchar

- System 10 data types not yet supported

Server Class - generic - server definition

- Server definition via *sp_servertype*:

  server_name - local Omni server name

  class - use *generic* for Generic Access Modules

  interface - always *oclient*

  network_info - name of server as found in *interfaces* file

Server Class - generic - table mapping

- Mapping done via *sp_addtabledef*:

  table_name - local Omni table name

  remote_name - path to remote table or view in the form:

  server.database.owner.object

  where:

  server - name of Omni server
  database - name of database in Generic Access Module (optional)
  owner - name of owner of remote object
  object - name of either table or view in server

Server Classes - Passthrough Mode

- Passthrough mode allows an Omni client to interact directly with a remote server. Once in passthrough mode, Omni simply 'passes through' client requests (without translation) and, in return, 'passes through' server result sets.

- To enter passthrough mode:

```
connect to server_name
```

- *server_name* - name of server as found in sysservers

- To exit passthrough mode:

```
disconnect
```

Server Classes - sp_autoconnect

- *sp_autoconnect* places new user into passthru mode whenever user logs in to Omni:

```
sp_autoconnect server_name, true | false,
   [ login_name ]
```

- *server_name* - name of server as found in sysservers
- *true* | *false* - "true" enables autoconnect mode; "false" disables it
- *login_name* - name of user; can only be used by 'sa'; defaults to current user name.

Server Classes - *defgen* Utility

- Provides quick and easy definition of remote tables to Omni.
- Generates SQL script file which contains *sp_addtabledef* and *create existing table* statements for tables in remote server.
- Remote tables can be specified individually, by owner, by database or by server.
- Remote server name and class must be defined in Omni first.
- Default data type conversions are used; the script file can be edited to override defaults.

Server Classes - *defgen* example

- Define all tables owned by 'dbo' in database 'pubs' in remote SQL Server:

```
defgen -Usa -P -SOMNI -Dtestdb -sSYBASE -dpubs
      -ndbo -Fsybpubs.sql
```

- Define all tables owned by user 'scott' in remote Oracle server:

```
defgen -Usa -P -SOMNI -Dtestdb -sORACLE -nscott
      -Fora.sql
```

- Define selected tables in remote DB2 server:

```
defgen -Usa -P -SOMNI -Ddb2_tables -sDB2
      -Fora.sql dsn8220.act dsn8220.emp
```

Server Classes - *defgen* example

- After *defgen* finishes, output file can be edited, if necessary, to modify default data type mapping
- Output file is then passed to Omni for execution:

```
isql -Usa -P -SOMNI < sybpubs.sql >sybputs.out
```

Support Issues

- If Omni is failing to connect to a remote server then try connecting direct to the remote server (without Omni).
- Remote tables can be altered outside of Omni. Hence, undefined results can occur if Omni's knowledge of external tables is not kept up to date.
- Permissions set via the *grant* statement are only local to Omni.
- Omni (current user) needs enough permissions to query remote catalog tables in order to perform a *create existing* command.

Navigation Server Product Overview

Navigation Server™
Product Overview

S Y B A S E®

Client/Server Architecture for the On-Line Enterprise

Navigation Server™

High Performance Database Management
for
High Capacity Database Requirements

Developed by NCR and Sybase

Major Features

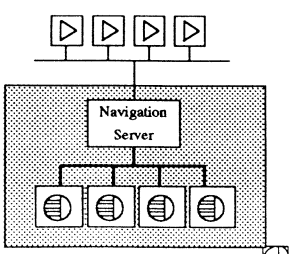
- Scalable Performance
- Transparent Access
- High Availability
- Leverages Parallel Computing Platforms
- Graphical Administration Interfaces
- Platform Independence

Performance

- Parallel Access to Partitioned Data
- Scaleup
  More transactions on greater database size
- Speedup
  Greatly reduced response time on large tables

Transparent Access

- One Logical Schema
  Application Programmer
  End User
- Appears as Single SQL Server
- Same Sybase Interfaces
- Interoperates with Sybase System 10 Products



Target Environments

- OLTP beyond 1000 tps
- DSS (Analytic Processing)
- DSS (Information Retrieval)
- Batch Processing
- OLCP (On-Line Complex Processing)

Very Large Database
Single Platform

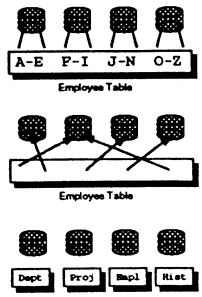
Paul Bass

Navigation Server Product Overview

Data Partioning and Parallelism

- Horizontal Partitioning of Tables
- Partitioning Choices
- Global Directory
- Partitions Assigned to SQL Servers
- Translation of SQL to Parallel SQL
- Parallel Access

Partitioning Choices



Partitioning Choices

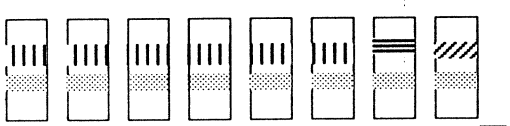
- Range
  - Value Ordered Requests
  - TPC/A or TPC/B type transactions
- Hash
  - Even Workload
  - Access to Entire Table
- Schema
  - Small Tables Not Partitioned

Future Partitioning Choices

- Replicated Small Tables
  - Join Performance
- Vertical (column sets)
  - Mixed Data Types - Blobs, Images
  - Physically Distributed Partitions
- Time Related
  - Mixed Data Storage Media

Partitions - Homes - Nodes

- Table Partitioned Across All or Subset of Nodes
- Collection of Nodes for Table = Home



Defining Partitions

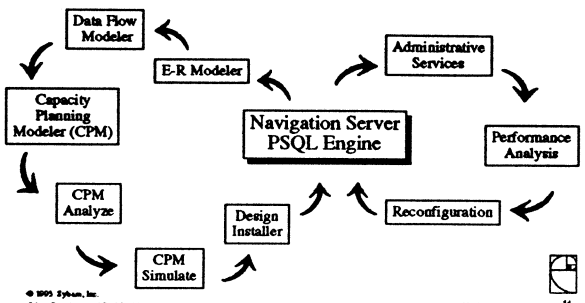
- Navigation Server Controls the Placement
- Calculated
  - via Configurator™
- Default
  - default scheme in the Global Directory

Navigation Server Product Overview

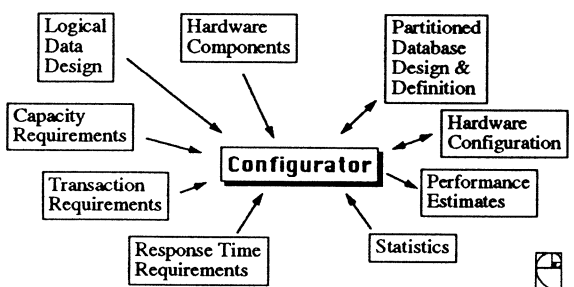
Global Directory

- Private Database
- Repository Necessary to Run the System
- Configuration Data
- Partitioning Information
- Access Via Navigation Server Administrative Services (NSAS)
- Controlled by Schema Server
- Always Mirrored

Navigation Server Lifecycle



The Configurator™

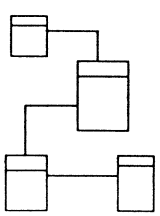


Configurator™ Uses

- Response to RFPs
  - Configurations
  - Performance Simulation
- Applications & Database Definition
- System & Database Installation
- System Tuning & Reconfiguration
- System Upgrades
- Capacity Planning
- Effects of Upgrade $$

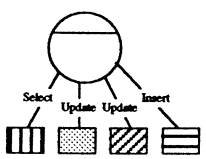
Entity Relationship Modeling (ERM)

- Logical Data Modeling
  - Tables - Rows - Indexes
- Table Capacity Estimates
- Based on Deft Tool
- GUI or DDL



Data Flow Modeling (DFM)

- Define Transactions
- Define Queries
- GUI or Transact SQL
- Extensions
  - # of Sources
  - Throughput Requirements
  - Response Time Requirements
  - Rates per Intervals



Navigation Server Product Overview

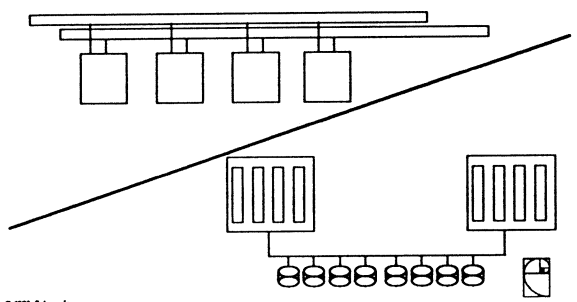
Capacity Planning Modeling (CPM)

- Generates hardware, software, and partitioning based upon ERM and DFM
- Analyzes model of the workload(s)
- Simulates the workload(s)
- Distribute - determine new partitioning based on existing workload
- Performance Reports
- Performance Graphics

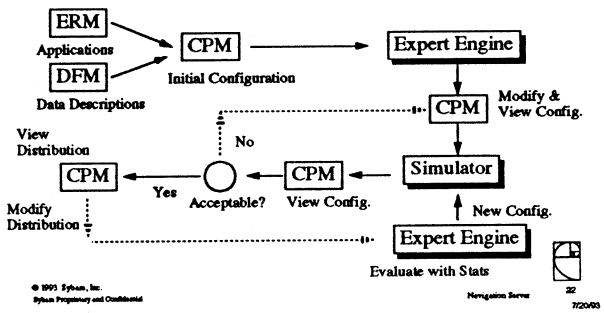
Configuration Generation

- Select Application(s)
- Choose Constraints
  CPU, Disk Capacity, Interconnect, Memory, Mirroring
- Choose Table Growth %
- Choose Transaction Rate Growth %

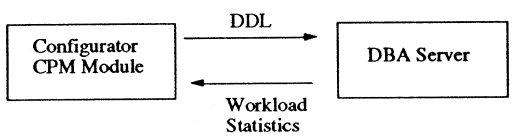
Configuration Output



Configurator Flow



Configurator Integration

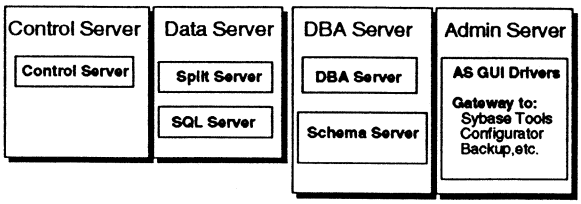


Other Database Defintions

- DBA Can Define "Home"
- No Partitioning Desired
- Quick Alterations Not Affecting Partitioning
- Extended Stored Procedure Facility in Admin. Svcs.

Navigation Server Product Overview

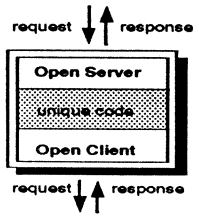
Basic Components



Communicate Via Message Passing: Shared Nothing

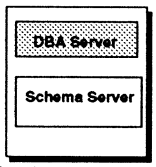
Component Structure

- Taking Advantage of Sybase Open Architecture



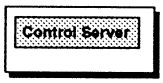
DBA Server

- Executes Parallel SQL Compiler
- DDL Interpreter
- Global Directory Functions
- Recovery Management - global deadlock detection
- Mirrored on Alternate Node (LCMP platform)
- Multiple DBA Servers
- Initiates Backup/Restore
- Schema Server Holds Global Directory



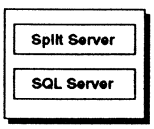
Control Server

- Receives Requests From Clients
- Initiates Parallel SQL Execution
- Sends Parallel Requests to Split Servers and SQL Servers
- Monitors and Initiates Recovery for Data Servers



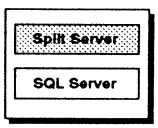
Data Server

- Unit of Parallelism
- Control Server Handles Recovery for Unit
- SQL Server Can Be VSA or Uni



Split Server

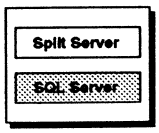
- Redistributes & Replicates Tables for Joins
- Merges Results from Split Tables
- Creates & Destroys Temp. Tables



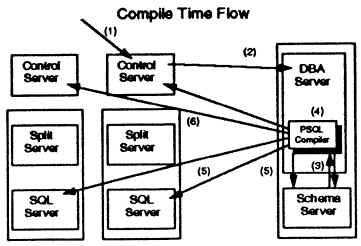
Navigation Server Product Overview

SQL Server

- Executes SQL Component of Parallel Plan
- uni-processor or VSA mode
- Standard SQL Server

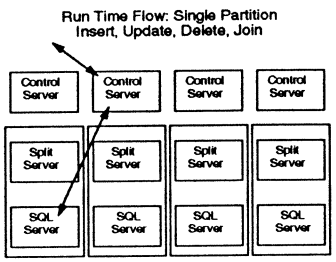


PSQL Compilation



(1) Request Received
(2) Passed to DBA Server
(3) Schema Server for Parsing
(4) Compilation and Optimization
(5) Stored Procedure Sent to All Servers Participating
(6) Loaded Into Memory for Execution

Single Partition Actions



(1) Request Received
(2) Request sent to single SQL Servers
(3) Results back to Control Server
(4) Results are returned to requesting client

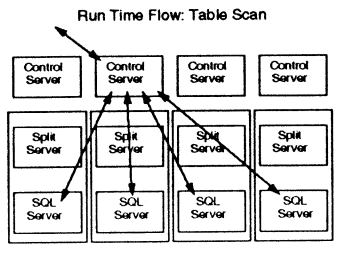
Multiple Partition updates handled with 2-phase commit

Multiple Control Servers Acting Concurrently

Transaction Processing Scaleup

- Nearly Linear for # of Partitions
- Assuming Transaction Load is Partition Balanced

Table Scan



(1) Request Received
(2) Stored Procedure Requests sent to all participating SQL Servers
(3) SQL Servers process stored procedure
(4) Control Server merges results and reorders if needed
(5) Results are returned to requesting client

Table Scan Speedup

- 25% overhead

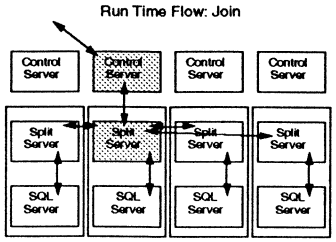
New time = Old time / (# of Data Servers * .75)

Old time of 60 minutes moved to Navigation Server with 16 partitions

60 / (16 * .75) = 5 minutes

Navigation Server Product Overview

Joins Crossing Partitions



(1) Request Received
(2) One Split Server becomes master
(3) Data is restributed as temporary tables
(4) In-place joins
(5) Results are merged
(6) Results are returned to requesting client

Types of Joins

- Redecluster 1
  Move rows from one table into home of other table
- Redecluster 2
  Move rows from both tables into a new home
- Replicate 1
  Replicate one table onto partitions of other tables home

Need Enough Work Space!

Complex Join Speedup

- 40% Overhead

New Time = Old Time / (# of DataServers * .60)
60 minutes with 16 partitions
60 / (16 * .60) = 6.25 minutes

Navigation Server Administrative Services (NSAS)

- Reduces Managing Complexities
  Hides Operating System
- Seven Applications - all GUI
  - Backup/Restore
  - Configuration Management
  - Designs
  - Stored Procedures
  - Logs
  - Tools

Backup/Restore

- Built Upon Sybase Dump/Load
- Single View of Data and Logs
- Parallel Execution
- Flexibility for Partition Restore
- Automatic or Time Specified
- Integrated With Install Design Facility

Greatly Reduced Backup Time!

Configuration Management

- Hardware & Software Configuration
- View of Software Mapped to Hardware
- Peripheral Mapping
- Database Topology
- Alerts & System Notifications
- System Maintenance Facilities

Navigation Server Product Overview

Designs

- Gateway to Configurator™
- Install Design Facility

Stored Procedures

- Similar to ISQL
- Auto Prompting for Parameters
- Results into Separate Windows
- Extensive On-Screen Help
- Accesses Global Directory

Log Management

- System/Navigation Server Statistics
- Error Logs
- Trace Logs
- Log to Unix files
- Display to Operator

Tools

- Gateway to isql
- Gateway to dbcc
  dbcc on individual partitions!
- Gateway to bcp

Application Qualification

- Very Large Single Store Database
- Maybe
  Large Number of Users
  Very High Transaction Rates
  Desire for "Open Solution"
  Need for Scalability
  Need for Flexibility

Navigation Server™

High Performance Database Management

for

High Capacity Database Requirements

Developed by NCR and Sybase

Kees Nieuwenhuijsen and Johannes Brouwer

AGENDA

1 PROBLEMS

2 THE DEMO CENTER

3 TECHNICAL OVERVIEW

4 DEMO-DATABASE AND DEMO-APPLICATION

5 TECHNICAL ISSUES

6 SUMMARY

Kees Nieuwenhuijsen and Johannes Brouwer

1 PROBLEMS

1.1 TWO SEPARATE ENVIRONMENTS

- Mainframe-environment
  - SNA-network
  - IBM-mainframe
  - large databases
  - character mode terminals

- UNIX- or OS/2-environment
  - LAN
  - UNIX- or OS/2-servers
  - local relational databases
  - PC's with GUI-applications

WISHES / DEMANDS

- Acceptance of both environments
- Transparent integration of various databases
- Maintaining investments
- Maintaining performance
- Use of modern user-interfaces
- Limit the uncontrolled growth of local databases

1.2 DOWNSIZING MAINFRAME TO UNIX

- Migrate complete application to UNIX
- Sudden switch from mainframe to UNIX too risky
- Preferably a more gradual migration

1.3 SYBASE SOLUTION

- Integration of the two environments
  - permanently
  - temporarily (gradual downsizing)
- Open Client / Open Server
- Connectivity products

2 THE DEMO CENTER

2.1 HISTORY

Sybase connectivity products are widely used in the USA (about 300 installations), but are relatively new to European customers.

Sybase-NL looked for partners to realize an environment to demonstrate their connectivity products.

2.2 CO-OPERATION

Various disciplines were needed to realize the demo-environment.

Five parties co-operated, each making a contribution to the demo-environment:

- SYBASE — rdbms
  connectivity products
- SAV — mainframe environment and expertise
  UNIX environment and expertise
- SUN — UNIX hardware
- UNIFACE — 4GL
- RCC — mainframe with DB2

3 TECHNICAL OVERVIEW

3.1 SYBASE CONNECTIVITY PRODUCTS

The Sybase CICS Gateway consists of four components:

- Net-Gateway
- Open Server for CICS
- Open Client for CICS
- SQL Gateway for DB2

Two additional components have been announced:

- Open Server for IMS-TM
- Open Client for IMS-TM

The Net-Gateway must be installed at the UNIX- or OS/2-platform.
One or more of the other components can be installed at the mainframe.

3.2 FIRST CONNECTION: UNIX TO CICS AT SAV

A connection was made within the SAV Computing Center at Leidschendam, where the UNIX-environment and the mainframe-environment where integrated.
Thus, mainframe-data could be accessed from UNIX.

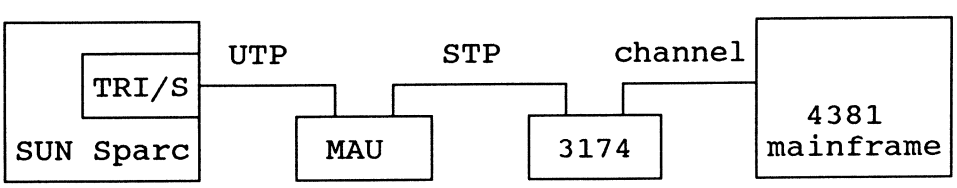
UNIX-environment: SUN Sparcstation 1+ in Ethernet LAN

-	operating system	SunOS
-	database	Sybase
-	network protocol	TCP/IP
-	PC's	Ambra 486
-	network software	PC/NFS
-	connectivity	Net-Gateway
		Sunlink SNA

Mainframe-environment: IBM 4381 in SNA network

-	operating system	MVS/SP
-	network software	VTAM
-	network protocol	SNA LU type 6.2
-	TP-monitor	CICS
-	database	VSAM files
-	terminals	3270
-	connectivity	Open Server for CICS

Physical connection:



TRI/S - SunLink Token Ring Interface/SBus
MAU - Multi_station Access Unit
3174 - IBM 3174 cluster controller with TR interface

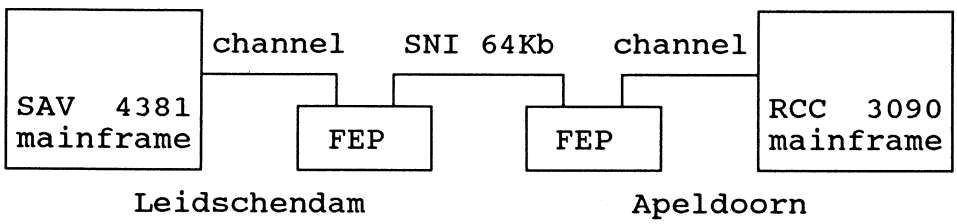
3.3 SECOND CONNECTION: UNIX TO DB2

The integrated environment at SAV was connected with the RCC-mainframe in Apeldoorn.
Thus, DB2-data could be accessed from UNIX.

Mainframe-environment: IBM 3090 in SNA network

-	operating system	MVS/ESA
-	network software	VTAM
-	network protocol	SNA LU type 6.2
-	TP-monitor	CICS
-	database	DB2
-	connectivity	Open Server for CICS Open Gateway for DB2

Physical connection:



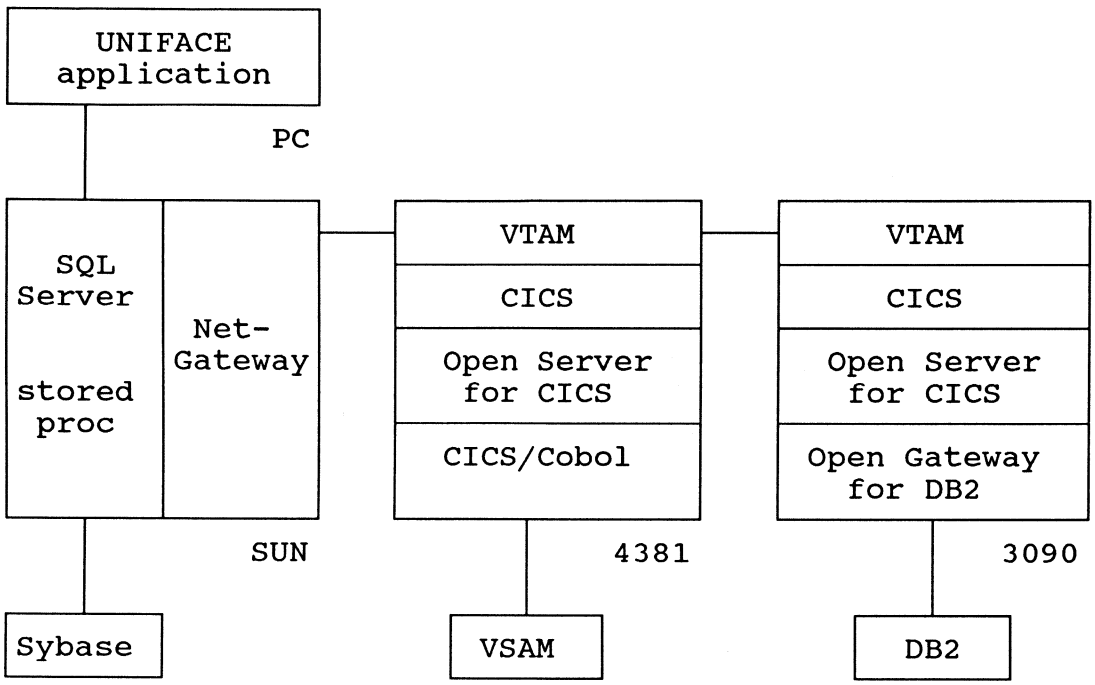
FEP - front-end-processor

3.4 THIRD CONNECTION: CICS TO UNIX

The Open Client for CICS was installed at the SAV-mainframe.
Thus, Sybase-data (under UNIX) could be accessed from the mainframe.

This connection could be realized using the existing physical connections.

3.5 ACCESSING VARIOUS DATA-SOURCES

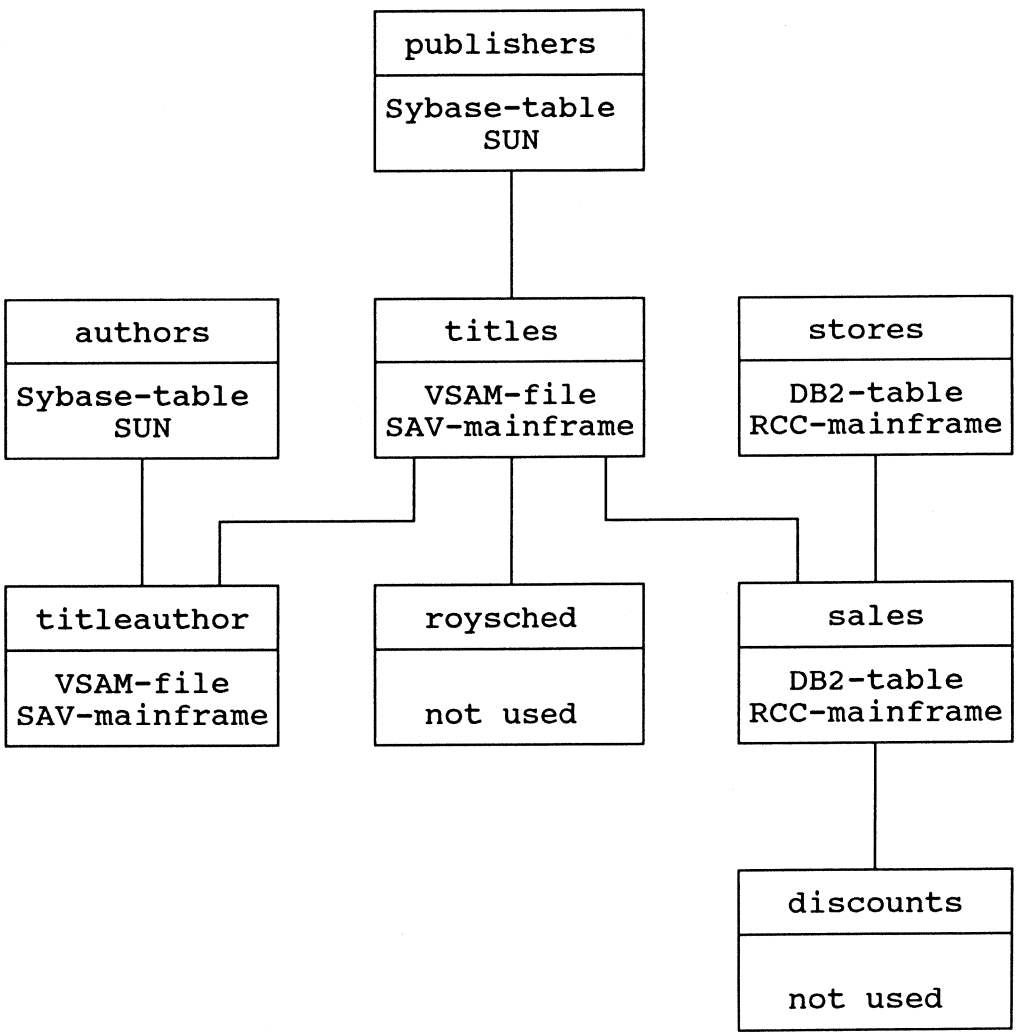


The Uniface application directs data requests to the Sybase SQL Server.
The Sybase SQL Server routes the request directly to a Sybase database, or to VTAM at the 4381 (SAV-mainframe).
VTAM routes the request to CICS: either CICS at the 4381, or (through VTAM) CICS at the 3090 (RCC-mainframe).
At the 4381, CICS activates a CICS/Cobol program that accesses a VSAM-file.
At the 3090, CICS routes SQL to a DB2-database.

4. DEMO-DATABASE AND DEMO-APPLICATION

4.1 DEMO-DATABASE

The connections are demonstrated using Sybase's PUBS database.
Six tables are used:



4.2 DEMO-APPLICATION

Three versions of the UNIFACE demo-application exist in three different platforms:

- UNIX (SUN Open Windows)
- PC (MS-DOS)
- PC (MS-Windows)

Functionality:

- retrieval and update of each entity
- retrieval of combinations of entities

example:

store	**7066**	**Barnum's**		DB2
		567 Pasadena Ave.		
		Tustin	**CA**	
sale		**A2976**		DB2
date		**5/24/87**		
quantity		**50**		
title		**PS8888**		VSAM
		Secrets of Silicon Valley		
author(s)		**427-17-2319** **Dull**	**Ann**	Sybase
		846-92-7186 **Hunter**	**Sheryl**	

5 TECHNICAL ISSUES

- Data distribution: central vs. local
- Security and control
- Two-phase-commit
  Technically possible, but rarely used
- Performance
- Uniface-specific issues:
  - re-writing stored procedures generated by Uniface
  - 'dummy' outer entity in external schemas

6 SUMMARY

During the conference, several demo-sessions have been scheduled at the SAV demo center at Leidschendam (near The Hague).
Attendees are invited to visit one of these sessions.

Server

Presentations

Distributed Client/server Design

by

John Kirkwood

Sybase
European User Group
Conference
Oct 1993

Abstract

This paper outlines a method to design distributed database systems. A brief look at fragmentation and allocation schemas is followed by a discussion of the changes to E/R modelling and functional analysis.

The problems of distributed update using two phase commit, remote procedure calls or replication are then looked at and the paper concludes with a discussion of the problems of applying a distributed design to the client/server architecture.

1 Introduction

This paper is the second in a series on distribution and client/server design which expands on the first paper in the areas of design methods and the problem areas of distributed update and client/server processing split. Which is not to say that these are the only or the most important problems as I would not minimise the difficulty of implementing schema changes in a distributed database or the general administration / management problems of such an architecture.

The topics which I shall be covering in detail are:

- a design method for fragmentation and allocation schemas
- a new approach to entity/relationship modelling
- an entity/function/location matrix for processing distribution
- approaches to distributed update control
- processing split in the client/server environment.

Before I get too far into the detail let me try to distinguish between distribution and client/server.

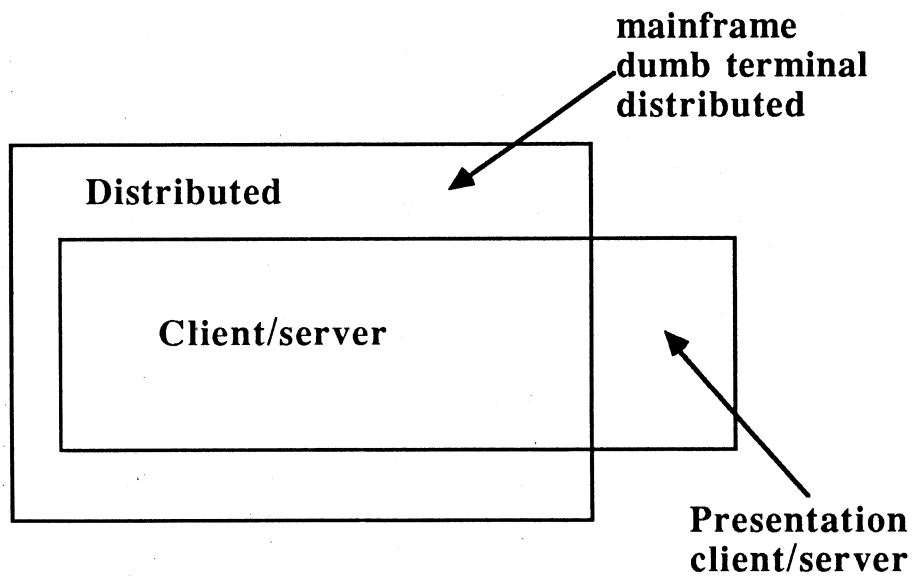


figure 1 levels of distribution

The systems which I shall be discussing are in the areas of figure 1 where client/server is a subset of distribution, which means that every hardware node is considered as an independent platform which is capable of executing application processing and storing data. Client/server architectures where the only consideration for the client is to remove screen manipulation and presentation from the server are not part of the discussion.

Distributed architectures where the desktop hardware is not capable of any processing or data storage are also not part of the discussion. Not that such systems are not valid or that they are not viable architectures, simply that the area of overlap between distributed and client/server allows a more general treatment which can then be modified to the limitations of these two special cases.

This is actually a very important point: distributed design is a physical design which involves many compromises based on the actual capabilities of each node in the network. In the global side of distribution it is valid to assume that each network node has the computing capability to support the processing and store the data, but in the more local client/server side it is often the case that the hardware at each network node has very different computing and storage capabilities. Our design method ignores these limitations but our physical implementation of the distribution must take them into account.

2 Fragmentation and allocation

(The first paper dealt with this in detail, so this is more of a summary.)

Our start point for distribution is the overall logical design which we must now distribute to the various network nodes based on our objectives. Such objectives will be totally installation and application dependent but they will usually include:

- maximise processing locality

- minimise network traffic

- minimise remote update

which means that we need to decide how to split the data up into fragments and then where to place each fragment.

This is illustrated in figure 2.

2.1 Fragmentation

This is not easy. It is reasonable to assume that we know the distribution sites. However, to determine how to split the data between these sites we need to know what data is accessed at the site - rows and columns - and how the data is accessed - read and/or write.

Why? As an example, with a simple 2 site distribution we will split horizontally if site_1 accesses half the rows and site_2 the other half but we will split vertically if both sites access all of the data but site_1 accesses half the columns and site_2 the other half.

In SQL terms:

horizontal	fragment_1	select * from customer where country = 'Scotland'
	fragment_2	select * from customer where country = 'England'

vertical	fragment_1	select cust_id, name, address from customer
	fragment_2	select cust_id, credit_limit, account_code from customer

There is, of course, a third type of fragmentation - mixed fragment - but it is simply a combination of these two.

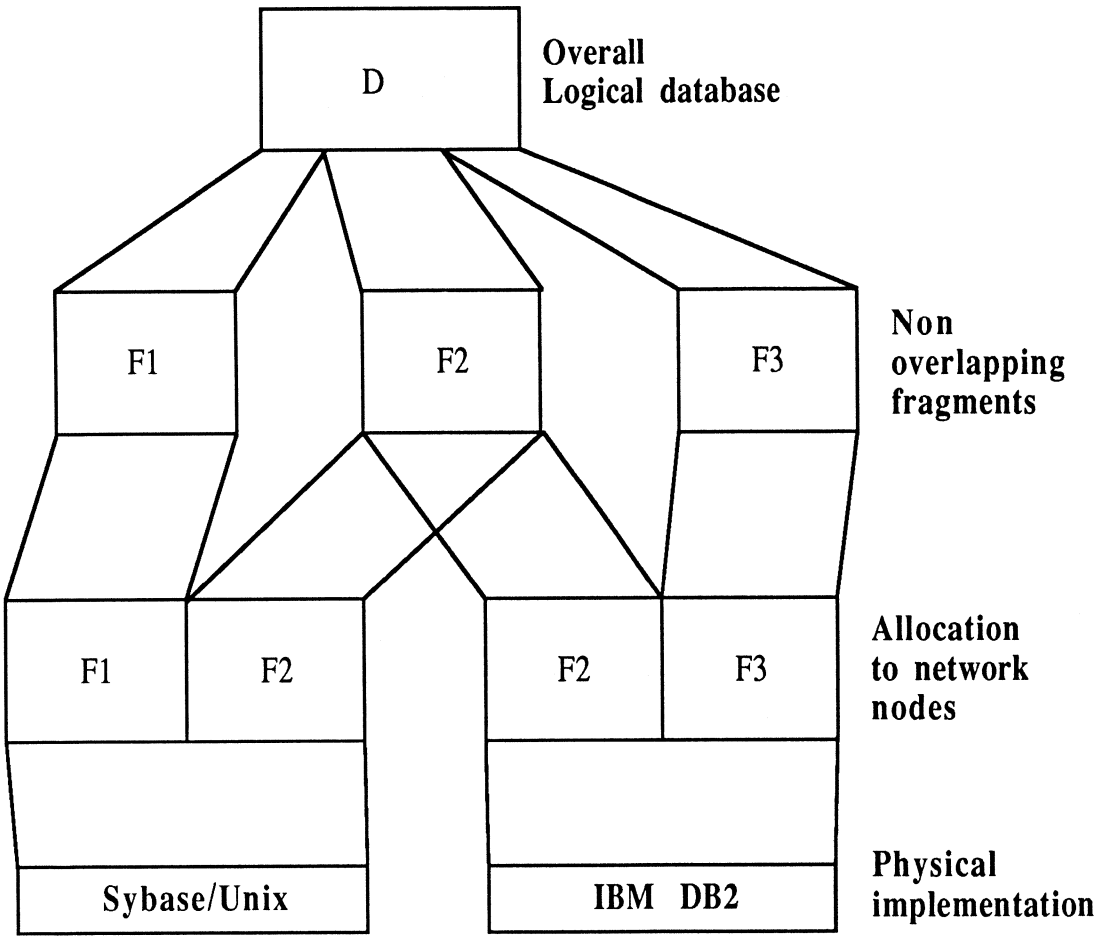


figure 2 fragmentation and allocation schemas

2.2 Allocation

The principal consideration for the design of fragment allocation is whether redundant allocation of fragments is allowed. Clearly when redundancy is allowed the network overhead of reads may be reduced but the update overhead is increased as we need to keep the replicates in step. The degree to which we keep the replicates in step with each other is very important to system performance and obviously depends on how much

each site needs up to the minute data. For this discussion we shall assume immediate propagation of update to all replicates.

2.2.1 Non-redundant allocation

There are many factors in this but the most important is to measure the amount of local work done by placing a fragment at a specific site. For each fragment we calculate:

number of local references = sum over all processes of
(frequency of use of process *
number of accesses per process)

This gives us a measure to determine which site is the most advantageous for each fragment and, as a side effect, indicates the amount of network traffic required when the fragment is not local to the process.

2.2.2 Redundant allocation

Again a simple approach is to measure the advantage to read processes by having a replicate of a fragment local and to subtract from this the disadvantage of having to update the replicate. This method does not take into account the saving of eliminating the remote read accesses - I leave this to you.

For each replicate at a site we calculate:

number of local references = sum over all processes of
(frequency of use of read processes *
number of reads by process)

less a "constant factor" times
(frequency of update from other sites *
number of updates by process)

The "constant factor" is a measure of the cost of a read versus the cost of an update, to take account of the additional space allocation, index and log accesses.

(Similar calculations may be made for vertical fragmentation which I have not included in this discussion. Mainly because there is not the time and it is more complicated as replication of primary key is always present which separates updates from deletes and inserts.)

3 New approach to entity/relationship modelling

Going back a little in the development cycle it is useful to see how the distributed requirement to split the processing so that it is located in the optimum location, imposes on our design method. This split is not superimposed easily on our existing methods which have a quite distinct approach to defining the functions and the data.

I will deal with the functions later but what we need to understand is that the logical data model stage requires a breakdown of the functions associated with the data values and their relationships. Therefore in the data model we need to define:

- domains: standard datatypes, complex datatypes
- data attributes
- validation rules and default values for the data attributes
- entities by amalgamation of the attributes
- foreign key referential integrity between the attributes
- other referential integrity such as control totals/denormalisation
- other business rules.

This is not too difficult to come to terms with in a Sybase environment where we are already defining this information in the data dictionary as user datatypes, rules, defaults, triggers and procedures. However as the other vendors are just beginning to catch up we need to impose this type of processing breakdown on the design methodology when we define the data model. What we are also doing is fixing the location of this processing definition at the server which runs the database software. We shall discuss the problems of this soon when we look more closely at the client/server implementation.

When you take this approach as distinct steps in a methodology what you find is that the data model definition now contains the processing associated with the inherent properties of the data and the relationships between them. The processing that is left to the functional analysis is the processing which is responsible for changing the state of the data. What I have found in applying this is that the objects of the system design begin to appear through the association of data characteristics and properties in the data model. This is not yet a firm approach to objects and referential integrity is still a problem but I think that it is the beginnings of a practical approach to object oriented database design.

4 Processing distribution

Having identified the functional components either as part of the data model or as external data manipulation components we need to decide where they will execute. In practice we have already identified the location of the internal functions: they reside with the data as they are part of the data model. However this applies only to the global view and the client/server architecture makes a distinction between definition and execution location. What we are looking at here is the execution location.

This is a two step process as we need to consider the global distribution first and then the client/server distribution within that. As stated before it is a physical design as we are attempting to place the functions to best satisfy our objectives and we need to consider the requirement of each category of user which carries out the same functions.

A useful approach is to create an entity/function matrix by:

- defining the entities which are accessed
- defining the processing on these entities
- create an entity/function matrix
- for each matrix entry determine the optimum execution location.

Let me illustrate this with an order entry example.

	customer	orders	stock	price terms	to follow
create	X	X			
update	X	X	X		X
credit check	X	X		X	
current function location	head office	entry clerk	warehouse	head office	entry clerk

Head office has a mainframe; the warehouse has a small Unix machine and the order entry function is on a Novell network. Just to state that each area has sufficient computing capability.

The data placement is:

customer	most used data replicated on the LAN; full file at head office
orders	daily orders on the LAN; full file at head office; to follow file at warehouse

stock	warehouse
price terms	head office; read only replicate on LAN

Based on this data placement we now superimpose our locations onto the matrix.

	customer	orders	stock	price terms	to follow
	LAN(extract) M/F	LAN M/F	W/H	LAN(copy) M/F	W/H
create	LAN	LAN			
update	M/F	M/F	W/H		W/H
credit check	LAN M/F	M/F		M/F	

There are obvious problems with this as we need a means of transferring orders from LAN to M/F; to control the update of the customer replicate at the LAN; to control the distributed update function and to decide where to execute the credit check. This is, of course, only one user category and the others are required to get the complete picture before we can make final decisions. However money where mouth is:

- make LAN customer file read only. Frequency of new customers is small and the insert can be routed to the mainframe and replicated to the LAN. When the network is down you can still accept the order.

- credit check on mainframe. It is a good example of business processing - may already be defined in the E/R model - and can be executed by rpc. Alternative is an order control file on the LAN as read only for the credit check.

- I doubt that there is much that can be done about the update but it does not need to be a two phase commit and I would execute a rpc to the mainframe.

This has been brief and not rigorous but I hope that you see the importance of a new approach to distributed design.

5 Control of updates

5.1 Two phase commit

A design consideration is to make all updates local and contained in the one node ie not distributed. However we cannot guarantee such a scenario and we must consider how we will deal with a transaction which requires to update data at more than one node in the network and for which the update at one node must not occur without the updates at the other nodes occurring. In a non-distributed update we insist on a multiple record update transaction being an all-or-nothing event ie all records are updated or no records are updated. In addition the locking mechanism which achieves this does not allow any changes to be seen by other transactions until all updates have been committed.

When we try to enforce this in a two phase commit we get the other characteristic of two phase commit: it is **slow**.

Let's look at a three node update as in figure 3.

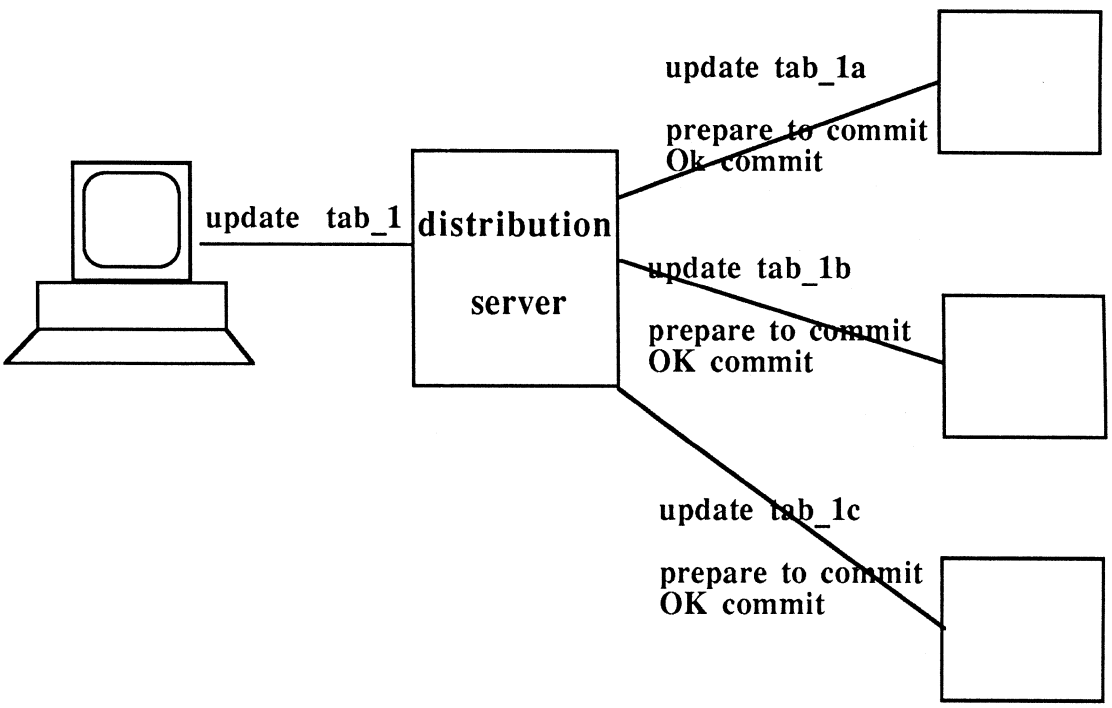


figure 3 two phase commit update

I know that they do not all work from a distribution server but it is easier to visualise. And it's a parallel two phase commit.

Now instead of a server transaction which has two network accesses: request and response, we have 6 network accesses per update: request/response, prepare/response, commit/response. To achieve full transaction control none of the data can be released until all commits have responded in which case there is another network access of

release/response. In the parallel two phase commit there is a reasonable chance that you will get away with a 2 times increase in response time but in the serial two phase commit it is more likely to be a 4 times increase.

Purely on a performance basis you must seriously evaluate the feasibility of two phase commit in a transaction based system.

However there are other aspects to two phase commit which have design ramifications, the most important being what the system does when failure occurs. There are three failure scenarios to two phase commit.

5.1.1 all or nothing

The problem with this approach is that there will always be some scenario which leaves the transaction in an inconsistent state which will require manual (DBA) intervention (to release locks).

For example: commit A, commit B, network failure C. Rollback B, network failure A. We cannot complete the transaction and we may have locks at A and C which need to be set off and the overall transaction is inconsistent unless we manually rollback A.

5.1.2 limited/no rollback

For these reasons most of the two phase commits - all that I know of - simply queue the commits for nodes which cannot be reached. Oracle has the concept of a primary node (not necessarily the control node) which must be committed. Usually the first to reduce the impact of queues when the complete network is down. When the failed node comes back on-line, the two phase commit software catches it up from the queue of updates.

This is fine if the failed node has not been used for update. However if local updates have been applied while the network has been unavailable for the remote updates, the data may have moved on and the referential integrity of the queued updates may force them to rollback. But you cannot rollback now as the other nodes committed and have had subsequent updates applied. Again a situation where manual intervention may be necessary.

So even the most sophisticated and fast two phase commits are not 100% foolproof and therefore you should consider an alternative.

5.2 Remote procedure calls

This may be serial or parallel - similar to two phase commit - but now we have the ability to work asynchronously and respond to the user after the first procedure call. This is really an optimistic approach where we take the 'no rollback' approach, waiting for failed nodes to become available, but in an asynchronous situation so that the response to the user is equivalent to a non-distributed update.

Of course we are now having to provide application routines to handle the failure of distributed nodes: at the simplest level by regular retry of procedure calls. Most of the database software will combine both worlds if necessary and allow rpcs to be wrapped in a two phase commit transaction if absolutely necessary.

5.3 Replication

Not all update situations require two phase commit or even an rpc style update scenario. These are reserved for the case where a single transaction must update data at more than one node. There is also the case where the single node update must be replicated to copies of the data at other network nodes.

In this situation there is no need for two phase commit and we can consider that all updating of the replicates will be asynchronous.

There are two considerations with replicates: the timing of the refresh - immediate or batch - and the manner of the refresh - physical or logical. The timing aspect is application dependent and depends simply on how long you can allow the replicates to be out of step with the primary data source.

The real design issue is how to refresh the replicates. You may have no choice of course, as the database software will do it in a specific fashion eg Oracle as physical, Sybase as logical.

The Sybase replication server is a logical replication as it fires the transactions from primary to replicate. At the simplest level this repeats the transaction at the replicate: at the most complex level it recognises differences and acts accordingly eg an insert becomes an update. However it is fraught with danger to allow local update to the replicate and logical differences will require application input and - more than likely - manual input to resolve problems.

If you do not permit local update of replicates or are willing to resolve the problems then the logical replication has much to recommend it.

If all you are doing is replicating look-up tables then complete overwrite is fine; otherwise think of replicating the transactions. As always with distribution it is application dependent and often a mixture is the best solution.

6 Client/server considerations

So far we have had no problems: we have been able to assume that the computing facilities at each site are able to execute the processing and to store the data under the control of the DBMS. This is reasonable to assume for global distribution - although we may need to cope with a heterogeneous environment - but when we look at each site there is a further level of distribution between client and server where the independent nodes do not have the same computing capability.

We now have to make a more detailed breakdown of the processing as we now have to differentiate between where the processing is defined and maintained and where it executes.

Table 1 shows the processing split, where it is defined and where it should be executed. The problems are with two aspects: domain integrity and non-foreign key referential integrity.

	Client/server processing	
Processing	Defined at	Executed at
domain integrity	server dictionary	client application
fkey referential integrity	server dictionary	automatic at server
other referential integrity	server dictionary	client application/server
business rules	server procedure	server
data access	server SQL	server
application processing	server/client	client

table 1 client/server processing split

3.1 Domain integrity

By this I mean the independent definition and validation of a field by a combination of column definition, rule and default. Obviously it is defined in the server dictionary but, just as obviously, it must be executed at the client as the data is input. Unfortunately this requires the DBMS dictionary to be replicated at the client which is not possible when the DBMS or the client does not support. A compromise must be made in this case by adding the domain integrity to the client validation with the control problems of keeping all processing replicates in step.

3.2 Referential integrity

Referential integrity splits into two major portions: foreign key validation and code value look-up. The foreign key validation is generally dealt with in the server dictionary and fires on insert/update/delete action: which is exactly what it should do and presents little problem. Look-up code field validation is a different matter as the data is quite correctly under server database control but, again it is an unnecessary network overhead to have to access the server every time a code field in input and there is a distributed case for look-up data being local in the client. (A look-up data fragment as complete tables replicated to the appropriate clients - an interesting little application of the Distribution theory. Notice that the problem of which data to replicate based on update activity is dealt with by the theory.)

This requires the client to be able to support the data, both with the software and with the storage capacity. The former requires replicated support for database fragments and for the client to be able to support the DBMS - or for heterogeneous replication support. Again it is a physical design problem and only you will know your application and environment well enough to determine how much data may be allocated to the client and how it will be accessed.

The same rules apply for the client/server distribution as apply to server Distribution, it's just that the physical environment in client/server if often not able to support the level of distribution that you require and compromises are necessary. But design your fragmentation and allocation schemas as if it were supported and then compromise - as always in physical design.

3.3 Transaction control

Within our client/server environment we now need to determine the optimum location for the application processing: server or client. Most of this is based on the minimise network traffic objective. Consider a client based transaction as in figure 4.

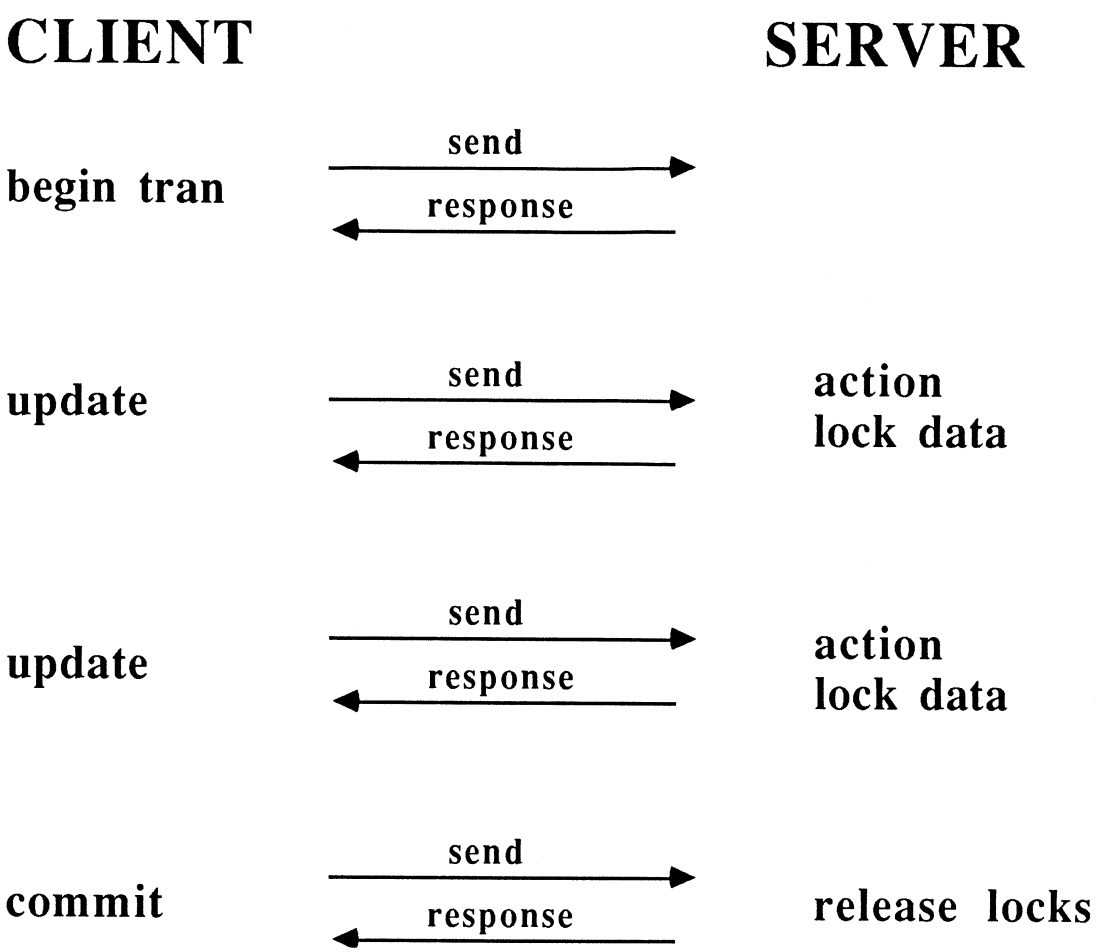


figure 4 client based transaction

With 8 network accesses we have probably increased the transaction time by a factor of 10 and with the length of time the locks are engaged, the overall throughput is likely to be severely affected. Our option is for server based transaction as in figure 5.

Figure 5 shows a procedure execution - which is my personal optimum - but there are occasions when procedures are not feasible and there is still a strong case for constructing the transaction as the client, sending it across the network to the server and executing it as a batch at the server. There is no more network traffic between client and server but there is an increase in the compilation time as the server has to interpret the SQL instead of it being a stored procedure.

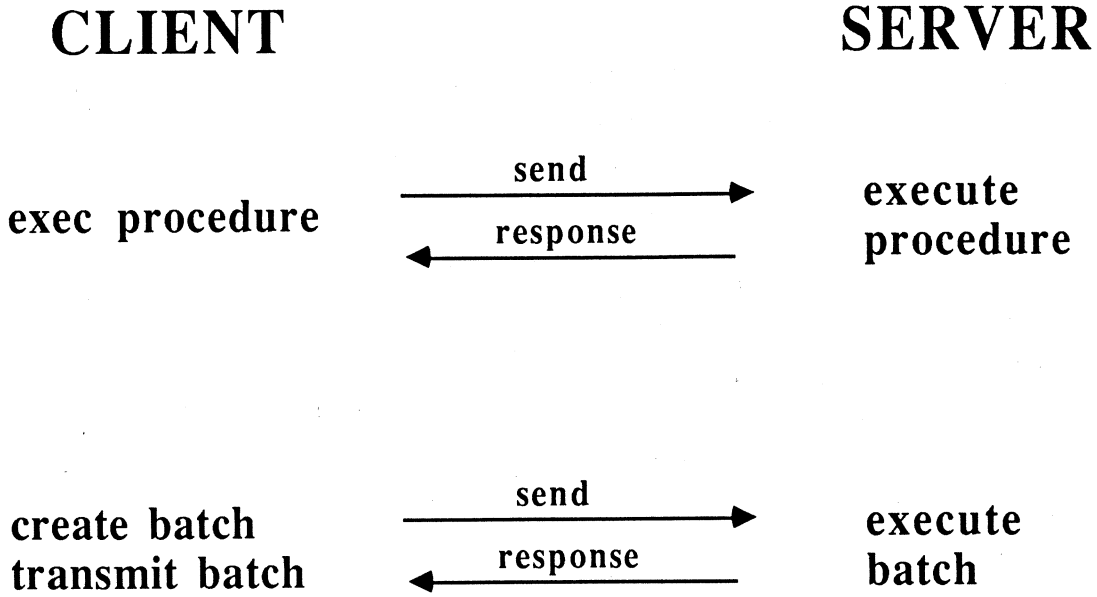


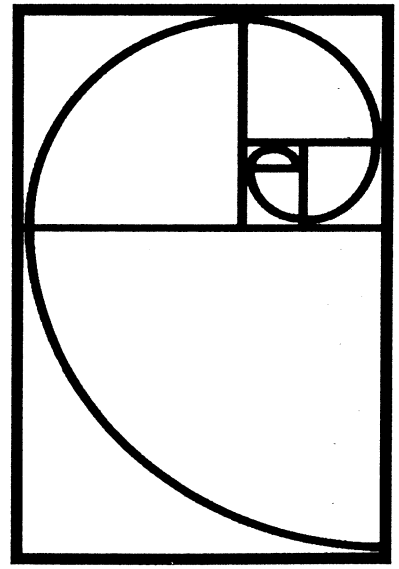
figure 5 server based transaction

7 Summary

This has been an introduction to distributed client/server design but I hope that I have persuaded you that a different approach is required to the data modelling and functional analysis stages. This achieves a logical distribution model which you must themntune to suit the physical implementation characteristics of the hardware and software platforms.

We need to take a more event driven approach to the design and add integrity and business rules processing to the data model. This allows us to concentrate on the external application logic which changes the state of the data and determine where this is best executed. This provides the global distribution and we then need to consider each node in this distributed architecture as an independent client/server environment to optimise the definition and execution of the functions. The client/server nodes require significant physical design to determine the execution locations of domain integrity, referential integrity and application transactions.

The Back-Up Server
DBA for R-10 of the SQL Server



SYBASE®

ClientServer Architecture for the On-Line Enterprise

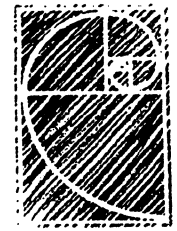
By Piet Burghardt

Overview

- **Open Server application shipped with SQL Server 10**
- **Designed to be a high-performance architecture for database and transaction log backup and restoration**
- **Addresses historical problems with dump and load**
  - *Poor Performance*
  - *Inconvenient Interface*
  - *Lack of Robustness*
  - *Lack of Device Independence*
- **Provides Sybase, its customers, and other vendors with an extensible architecture for the future**
  - *Published RPC API for the Backup Server*
  - *Published subroutine API for an auxiliary server I/O Library*

Performance Features

- **Improved Concurrency**
  - Supports concurrent dumps from the same or different SQL Servers.
  - Minimal synchronization of access to database pages between SQL Server and Backup Server. DUMP / LOAD can utilize full bandwidth of the I/O subsystem without being paced by SQL Server events.
- **DUMP to Multiple Devices Concurrently *(Striped Dumps)***
  - Up to 32 striped devices reducing DUMP time by a factor of N.
  - May eliminate need to change volumes allowing complete automation.
  - Striped DUMPs can cross different device types.
- **Enhanced support for more device types**
  - Supported types include: 8mm tape; DAT ; 9-track tape; streaming cartridge; disk file; raw (foreign mounted) disk; floppy (or other removable) disk.



Performance Features (cont.)

- **Dump to Remote Devices Across the Network**
  - Dump to devices of another Backup Server listed in INTERFACES.
  - Backup Server can run on a machine without a SQL Server (and can be licensed independently).
  - Flexibility to insure minimal impact to SQL Server throughput.

- **Multi-volume dumps and multi-dump volumes**
  - Continued support for spanning dumps across volumes. RPC Messaging will replace the console interface.
  - New support for multiple dumps within a single tape volume for devices supporting the EOT mark (double EOF). INIT option overrides.
  - ANSI standard volume labeling supported for individual LOADs from multi-dump volumes.

Administrative Features

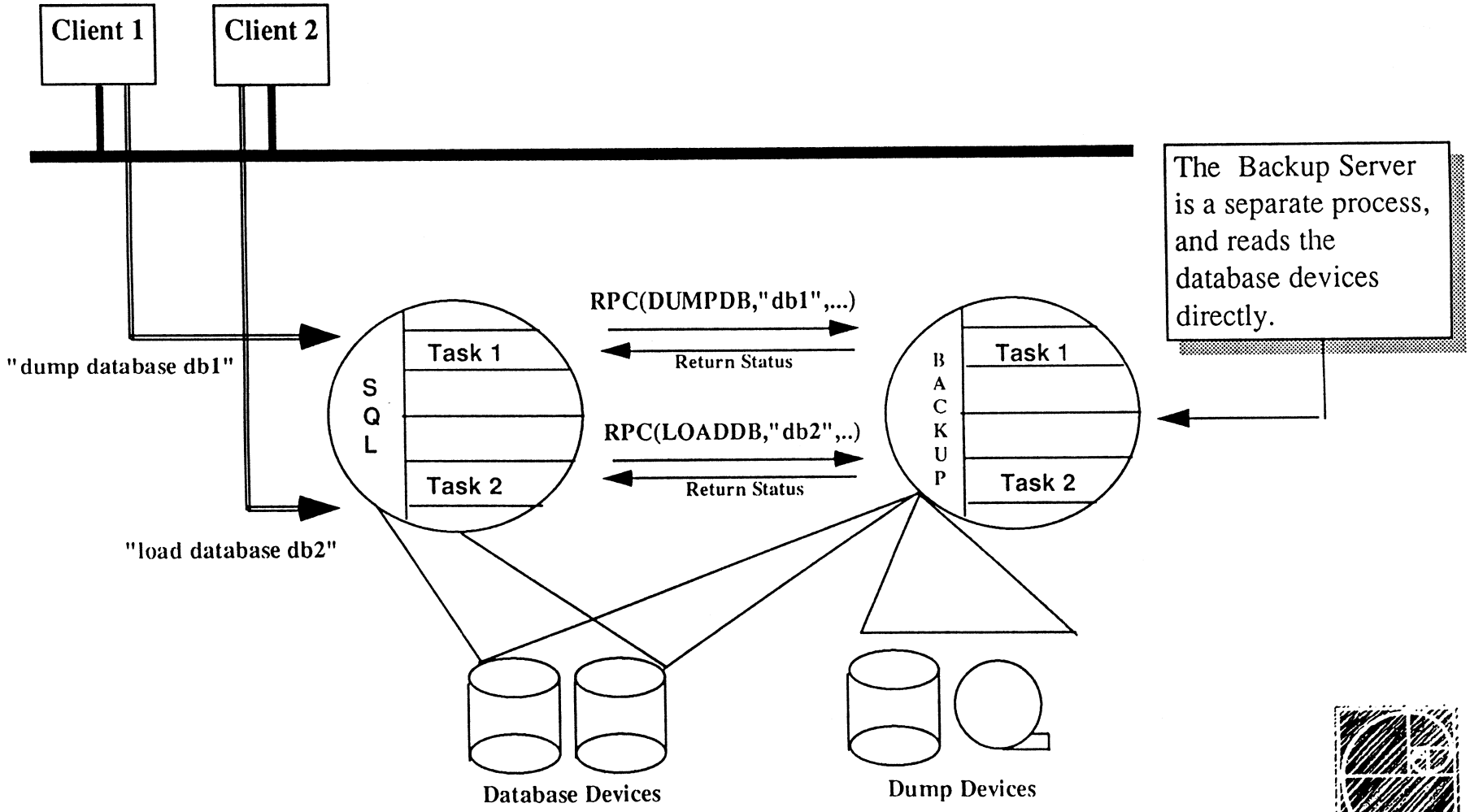
- **Console Program Replaced with RPC Messaging**
  - "Volume Needs Change" messages sent to either the client which issued the DUMP / LOAD or the Backup Server's Console.
  - Any *other* client workstation can coordinate volume mounts via any Open Client application capable of an EXECUTE with parameters.
- **Free Space Threshold on Log Device**
  - Free Space monitored in Server providing execution of a stored procedure to automatically DUMP & TRUNCATE Log
  - Eliminates risk of lost transactions from a DUMP with NO_LOG
- **Automatic Support for Device Types**
  - Backup Server "senses the control methods" for supported device types.
- **ANSI standard Tape Labeling**

Programming Features

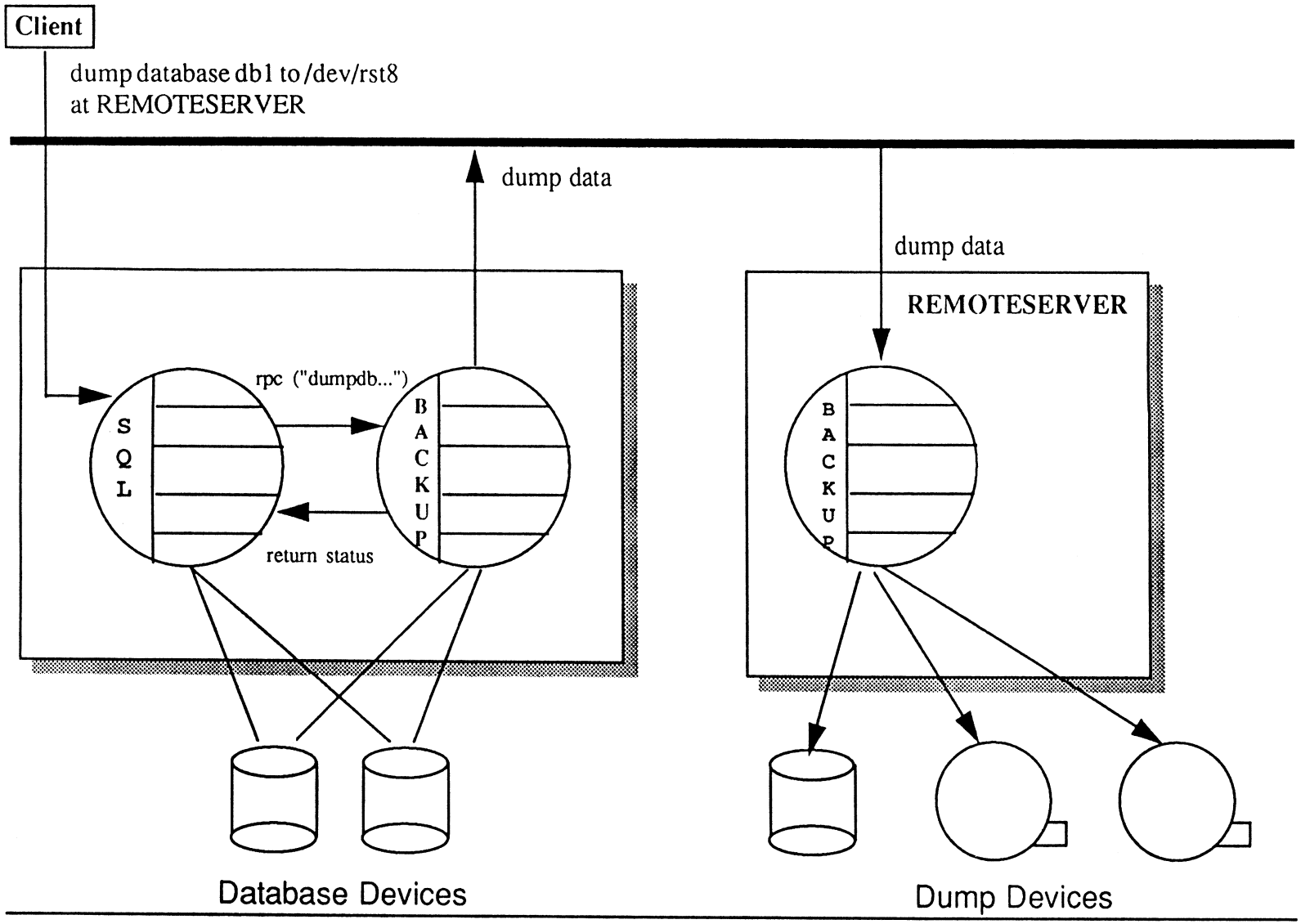
- **RPC API for Communication with Backup Server**
  - Provides informational messaging such as DUMP progress
  - Users / vendors can write their own Backup Server to exploit platform-specific features not addressed by Sybase.
  - Security imposed to insure **only** SQL Server's DUMP / LOAD can execute most RPC APIs.

- **Auxiliary Server I/O Services Library API**
  - Backup Server I/O built on abstract library of operations on block-buffered, removable-volume device objects.
  - Available to customers and vendors in a documented, self-contained, and reuseable layer above the Open Server API.
  - Allows customers and vendors to write auxiliary servers, based on Open Server, for specialized applications requiring high I/O performance.

Backup Server Architecture



Remote Backup and Recovery



Installing the Backup Server

- **Backup Server installed during SQL Server installation**
  - Resides on same distribution media
  - Installed into same directory
  - Has the same permissions as the SQL Server binary
  - Remote Backup Servers can be licensed & install separately
- **Configure SQL Server for Remote Access**
  - Necessary to enable RPCs between SQL Server & Backup Server
  - *sp_configure 'remote access', 1*
- ***sp_addserver* all Backup Servers**
  - *sp_addserver logical_svrname [,{local | null}] [, interfaces_name]*
  - Only 1 local server allowed per SQL Server due to the requirement of device-level access to the data. This is the Backup Server that is instantiated during startserver.
  - Installation defaults local Backup Server as SYB_BACKUP.
  - Requires SA authorization

Installing the Backup Server (cont.)

❑ Edit Interfaces Entries

- SQL Server machine requires entries for all "local" backup servers and any Remote Backup Servers to which dumps will be submitted:

```
SVR1:
        master tcp sun-ether paladin 4096
        query tcp sun-ether paladin 4096
SYB_BACKUP:
        master tcp sun-ether paladin 3110
        query tcp sun-ether paladin 3110
REMOTE_BACKUP:
        master tcp sun-ether gaia 3617
        query tcp sun-ether gaia 3617
```

- Remote Backup Servers require entries for each "local" Backup Server from which dumps originate

```
SYB_BACKUP:
        master tcp sun-ether gaia 3617
        query tcp sun-ether gaia 3617
SVR1_BACKUP:
        master tcp sun-ether paladin 3110
        query tcp sun-ether paladin 3110
```

Starting the Backup Server

- **Backup Server process automatically started by *startserver***
  - Backup Server started is designated by "local" in sysservers
  - Dump / Load does *not* auto load backupserver and will fail if the local backupserver is not yet running.
  - If Backup Server fails to load, *startserver* will provide option to either continue loading SQL Server only or fail both.
- **Backup Server process can be independently started**
  - If *startserver* failed or to boot a Remote Backup Server
  - Executable Command line to boot a Backup Server:
    backupserver [-C*user_connections*]
    [-S*backup_server_name*]
    [-I*interfaces_filename*]
    [-M*sybmultbuf_path*]
  - *startserver* with a Backup Server RunFile can also be used
    startserver [-f*runserver_filename*]
  - User starting process must have write access to dump devices

Configuration Issues

- **Additional Machine Memory**
  - Dependent on database size & dump device setup
  - Engineering will be developing an approximate formula and doing additional verification tests. This formula should be finalized early in the beta cycle.
- **User Connections (both SQL Server & Backup Server)**
  - Two connections per active DUMP
  - One connection per active LOAD
  - One connection per active DUMP or LOAD for volume messages.
- **Dump Devices**
  - Local Dump Devices no longer need to be listed in sysdevices. Users can choose between specifying logical device names (as found in sysdevices) or physical device names.

DUMP Operations

❑ Syntax of new DUMP command

DUMP {DATABASE | TRANSACTION} database_name
TO stripe_device [AT server_name] [DENSITY = d,] [BLOCKSIZE = b,] [CAPACITY = k,]
[user_option | user_option = value, ...]
[STRIPE ON
(stripe_device2 [AT server_name] [DENSITY = d,] [BLOCKSIZE = b,] [CAPACITY = k,]
[user_option | user_option = value, ...]
[, stripe_device3 [AT server_name] [DENSITY = d,] [BLOCKSIZE = b,] [CAPACITY = k,]
[user_option | user_option = value, ...])]
[WITH
{ NOTIFY = {"*CLIENT*" | "*OPERATOR_CONSOLE*"}
DUMPVOLUME = vol_name ,
NODISMOUNT | *DISMOUNT* ,
UNLOAD | *NOUNLOAD* ,
DEALLOC | *NODEALLOC* ,
RETAINDAYS = int_value ,
INIT | *NOINIT*,
TRUNCATE_ONLY | NO_LOG | NO_TRUNCATE
}] [user_option | user_option = value, ...]

LOAD Operations

❑ Syntax of new LOAD command

LOAD {DATABASE | TRANSACTION} database_name FROM
stripe_device [AT server_name] [DENSITY = d,] [BLOCKSIZE = b,]
[user_option | user_option = value, ...]
[WITH STRIPE ON
(stripe_device2 [AT server_name] [DENSITY = d,] [BLOCKSIZE = b,]
[user_option | user_option = value, ...]
[, stripe_device3 [AT server_name] [DENSITY = d,] [BLOCKSIZE = b,]
[user_option | user_option = value, ...])]
[WITH
{ NOTIFY = {"*CLIENT*" | "*OPERATOR_CONSOLE*"},
DUMPVOLUME = vol_name ,
FILE = tape_filename ,
HEADERONLY,
NODISMOUNT | *DISMOUNT* ,
UNLOAD | *NOUNLOAD* ,
DEALLOC | *NODEALLOC* ,
}] [user_option | user_option = value, ...]

New Syntax Definitions

- **WITH STRIPE ON (stripe_device [AT server_name],...)**
  - Up to 32 devices for a single DUMP / LOAD stripe set
  - Each allocation unit's pages divided across stripe set
  - Each volume saves its stripe set ordinal number. Used in LOAD and change of volume verification. Allows LOADs from a smaller number of devices than used during DUMP
- **WITH NOTIFY = {"*CLIENT*" | "*OPERATOR_CONSOLE*"}**
  - "CLIENT" is workstation who issued DUMP / LOAD (default on UNIX)
  - "OPERATOR_CONSOLE" is Backup Server's console or, if supported by the system, an "operator terminal" (default in VMS)
- **DENSITY = d**
  - On UNIX, density is determined by the device files' major/minor numbers
  - If not included, Backup Server uses OS' default
- **BLOCKSIZE = b**
  - Allows user to choose block size that yields good tape utilization and continuous streaming (tends to increase transfer efficiency to disk).
  - Backup Server will determine and choose this by default.

New Syntax Definitions (cont.)

- **CAPACITY = k**
  - Only necessary for those devices which can't detect EOT marker.
  - Used to signal volume change messages after *k* kilobytes written.
  - Depends on blocksize and density. Tech Support will have access to diagnostics that report Backup Server's blocksize and density choices
- **DUMPVOLUME = vol_name**
  - Checked against ANSI label volume name field.
  - On VMS, already done through VMS INITIALIZE command.
  - On UNIX, written in ANSI volume label for initial volume at DUMP time.
- **NODISMOUNT | *DISMOUNT***
  - Controls whether tape should remain mounted after completion.
- **UNLOAD | *NOUNLOAD***
  - Controls whether tape is rewound and unloaded after completion.
  - UNLOAD implies DISMOUNT.
  - UNLOAD should be used after the last DUMP in multi-file dumping.

New Syntax Definitions (cont.)

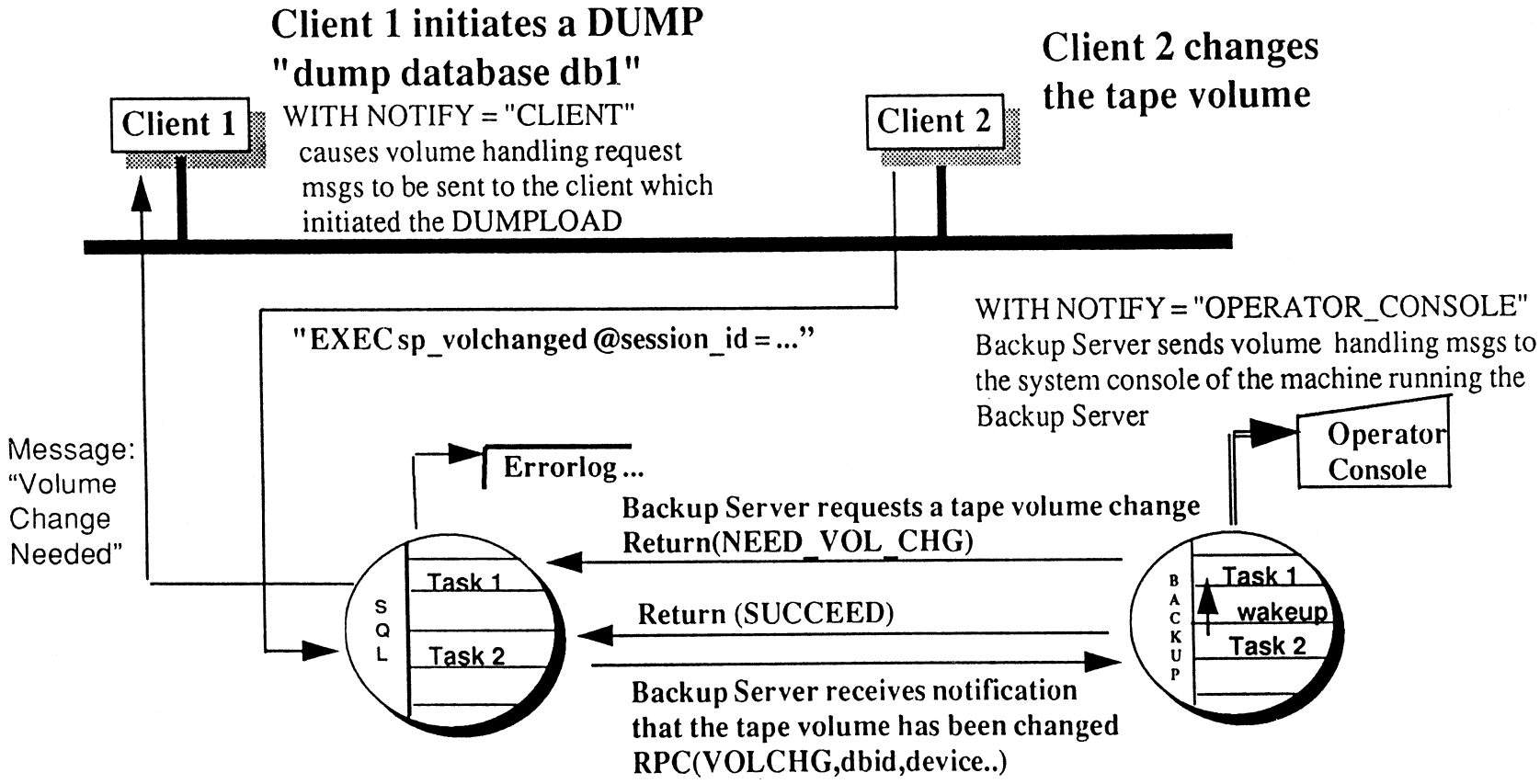
- **DEALLOC | *NODEALLOC***
  - Backup Server always allocates (exclusively) the tape device.
  - DEALLOC should be used after the last DUMP in multi-file dumping.
- **RETAINDAYS = d**
  - Number of days that must pass before tape can be overwritten
  - Returned errors can be ignored if desired
- **INIT | NOINIT**
  - INIT overwrites all previous contents.
- **FILE = tape_filename**
  - Used at LOAD time to advance tape to particular dump.
  - Tape file name is generated by Backup Server during the DUMP and returned in the SUCCEED message stream. It must be trapped and retained for LOADs from multi-dump volumes.
- **HEADERONLY**
  - Prevents LOAD
  - Returns dump header information from each volume in the stripe set

An Early Look at Performance

- **Test Environment**
  - 2-CPU Sun 4/690MP, 128Mb RAM, 6 SCSI-2 disk spindles (7th for log mirror) for data , 4 IPI Panther disk spindles for dumps connected to 2 controllers
    2 Exabyte 8mm Tape Drives connected to 1 SCSI-2 controller,
  - Run against TPC database during active TPC-B benchmark run

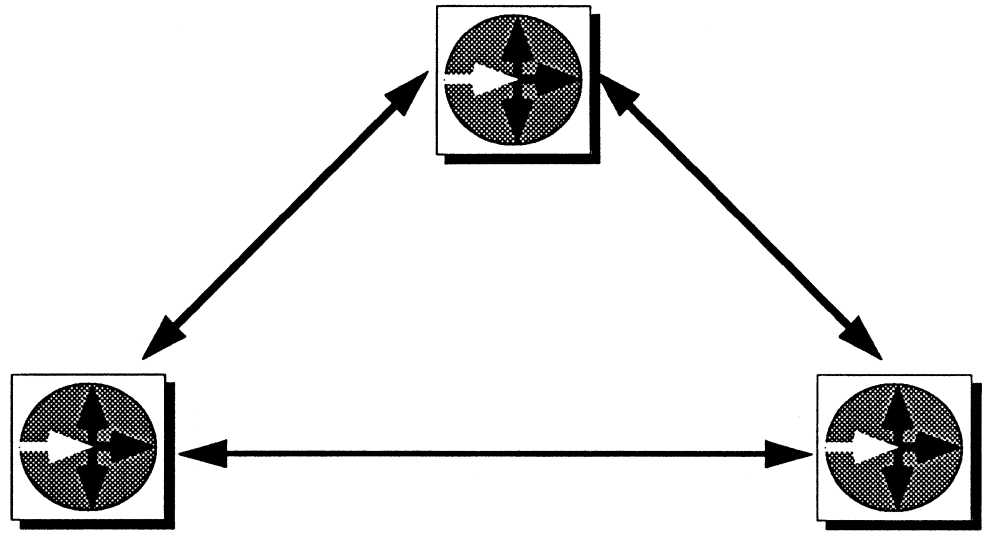
- **Results**
  - Improvement by a factor of 7.1 in DBMS throughput over v4.9.1 during a DUMP DATABASE
  - DUMP "currency" (the number of transactions captured in the dump divided the number that commit during the dump) improved by a factor of 8.5
  - Effective TPS rate during LOAD DATABASE increased 3900% from 0.6 TPS in v4.9.1 to 23.5 TPS with System 10
  - Highest DUMP rates were 6.4Gb per hour
  - Early testing showed 95% scalability in going from one 8mm tape drive to two 8mm tape drives

Messaging for New Tape Volumes



Notifying the Backup Server that the Tape Volume has been changed:

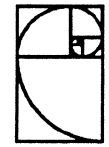
EXECUTE sp_volchanged
@session_id = **session_id,** @devname = **device_name [at server_name],**
@fname = **tape_filename,** @action = {**"proceed' | 'retry' | 'abort'**}



Sybase Replication Server Design Presentation

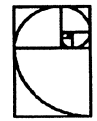
By Sachin Chawla

Client/Server for the online Enterprise



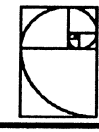
Advantages of Replication

- Improves Performance
  - Offloads Mainframe
  - Reduces Network and Data Server Loads
  - Users and Data Can be Co-located
  - User Transaction Response Time Reduced
  - Scaleable Architecture
- Improves Availability
  - Applications May Continue In Spite of Mainframe, Network and Single/Multiple Site Failures
- Supports the Organization
  - Partitioned Administration and Control
  - Transaction Processing vs Decision Support



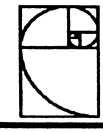
Applications Benefiting From Replication

- Geographically Distributed Mission Critical
  - Trading
  - Banking
  - Credit Card
  - Reservations Systems
- Decision Support
  - Local Applications
  - Distributed Applications Supporting Mission Critical



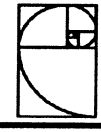
Technical Approaches To Replication

- Tight Consistency
- Loose Consistency



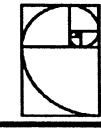
Tight Consistency Replication

- Typically Uses 2 Phase Commit
- All Copies Are Identical
- Replication Is Transparent To Applications
- High Overhead For Protocols
- May Reduce Fault Tolerance, Availability



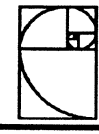
Two Phase Commit

- Used To Implement Tightly Consistent Distributed Data
- One Transaction Bears the Delay Due to Synchronous Effects of all Servers
- Will Fail if any Component of the System is Unavailable
- Does Not Scale Well - Adding Additional Distributed Components Severely Degrades Performance.
- Appropriate When Absolute Consistency is a Requirement. High Speed Reliable LAN Preferred.



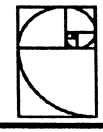
Loose Consistency Replication

- 'Primary' Copy Of Data
- Primary and Replicates Are Not Identical
- Replication Is Not Transparent To Applications
- High Performance - Low Overhead For Protocols
- High Data Availability and Resilience to Failure



Architectures for Loose Replication

- Tape Dump And Reload
- Table Snapshot Replication
- Sybase Online Asynchronous Replication

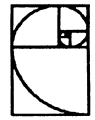


Tape Dump And Reload

- Database Backups
- Database Logs
- Selective Table Dumps
- Download Data To Remote Sites
- Upload Transactions To Central Site
- Typically Long Latency Period
- Large Volumes of Data Difficult

Table Snapshot Replication

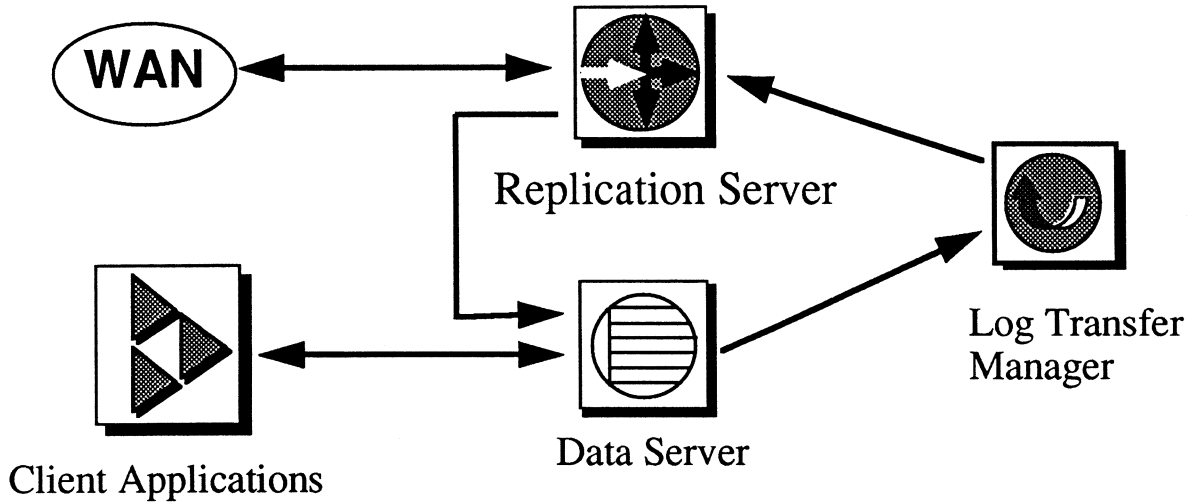
- Supported By Some Commercial DBMS
- Table Dump and Reload
- Replication Of File Changes
- May Allow Subsets Of Data
- Proprietary Solutions
- Typically Medium Latency
- Not Transaction Consistent
- No Provision For Remote Update



Sybase Online Asynchronous Replication

- Continuous Distribution Of Updates
- Row Level Granularity
- Latency Measured In Seconds
- Maintains Transaction Consistency
- Asynchronous Transaction Processing
- Heterogeneous Open Architecture
- Fault Tolerant Design

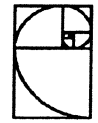
Replicated Data System Components



- Client Application Updates Primary Data
- Data Server Manages Data at Local Site
- Log Transfer Manager Notifies Replication Server of Primary Data Updates
- Replication Server Coordinates Data Replication with Local Data Server and Other Replication Servers

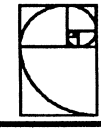
Connecting Components

- Components Use SYBASE Client/Server Interfaces (C/SI)
- Interfaces File at Each Site Defines Local/Remote Replication Servers and Data Servers
- A 'Connection' Defines a Replication Server Connection to a Database
- A 'Route' Defines a Path from One Replication Server to Another



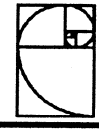
Process for Replicating Data

- Create Database Table at Primary Site, if it does not exist
- Create a Replication Definition to Describe Table
  - Column
  - Data Type
  - Primary Key for Row
  - Columns Used in a Subscription **where** Clause
  - Location of Primary Table
- Create an Empty Table where Data Will be Replicated
- Create Function Strings if Data Server is Not SYBASE SQL Server
- Create Subscriptions at Site where Data is to be Replicated
  - `create subscription affordable for NYSE_Stocks with replicate at Chicago.Stock_db where price >= $20 and Price <= $56`



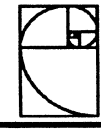
Replicating Stored Procedures

- Create Stored Procedures at Primary Site, if it Does Not Exist
- Create a Replicated User Function Definition
  - Procedure Name
  - Parameters and Data Types
  - Location of Primary Data
- Create Stored Procedures at Replicated Sites
- Create Function Strings if Data Server is not SYBASE SQL Server



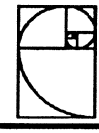
Heterogeneous Data Server Support

- SYBASE SQL Server Fully Supported
- Other Data Servers Required:
  - C/SI Support Through Data Server Directly or Through Open Server Gateway
  - Log Transfer Manager (LTM) Support
  - Error: Class Definitions, Mappings and Processing Actions
  - Function Strings and Function String Classes to Send Directives (i.e. .**insert, delete, update**) to Data Servers

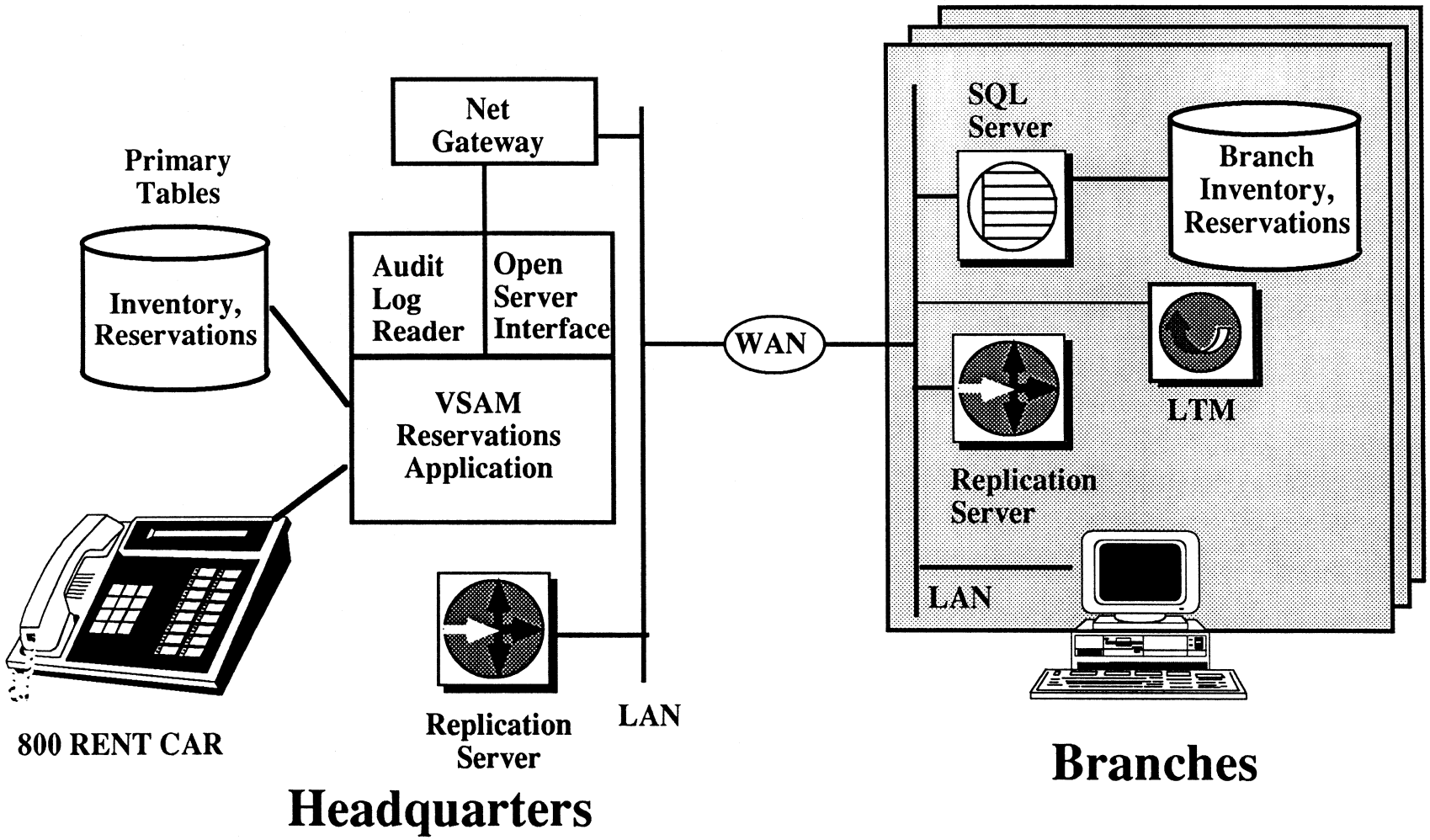


Example Application Scenario

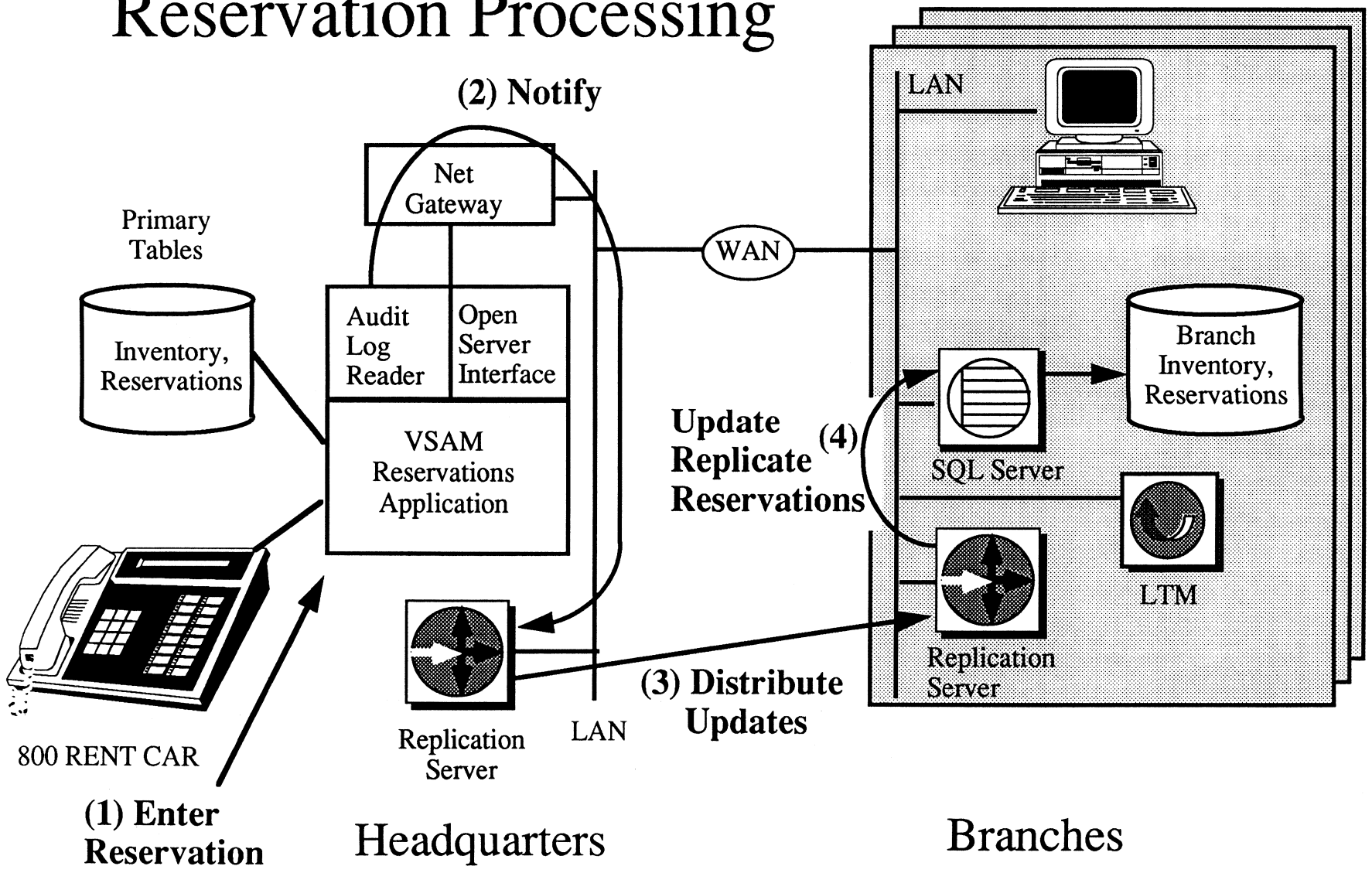
- Application: Rental Car
- Current System
  - Central Mainframe: CICS/VSAM
  - Problems: Old, Overloaded, Expensive, Failure-prone, Keep "Losing" Inventory
- Objectives of New System
  - Increase Performance
  - Increase Reliability
  - Fix Inventory Tracking Problems



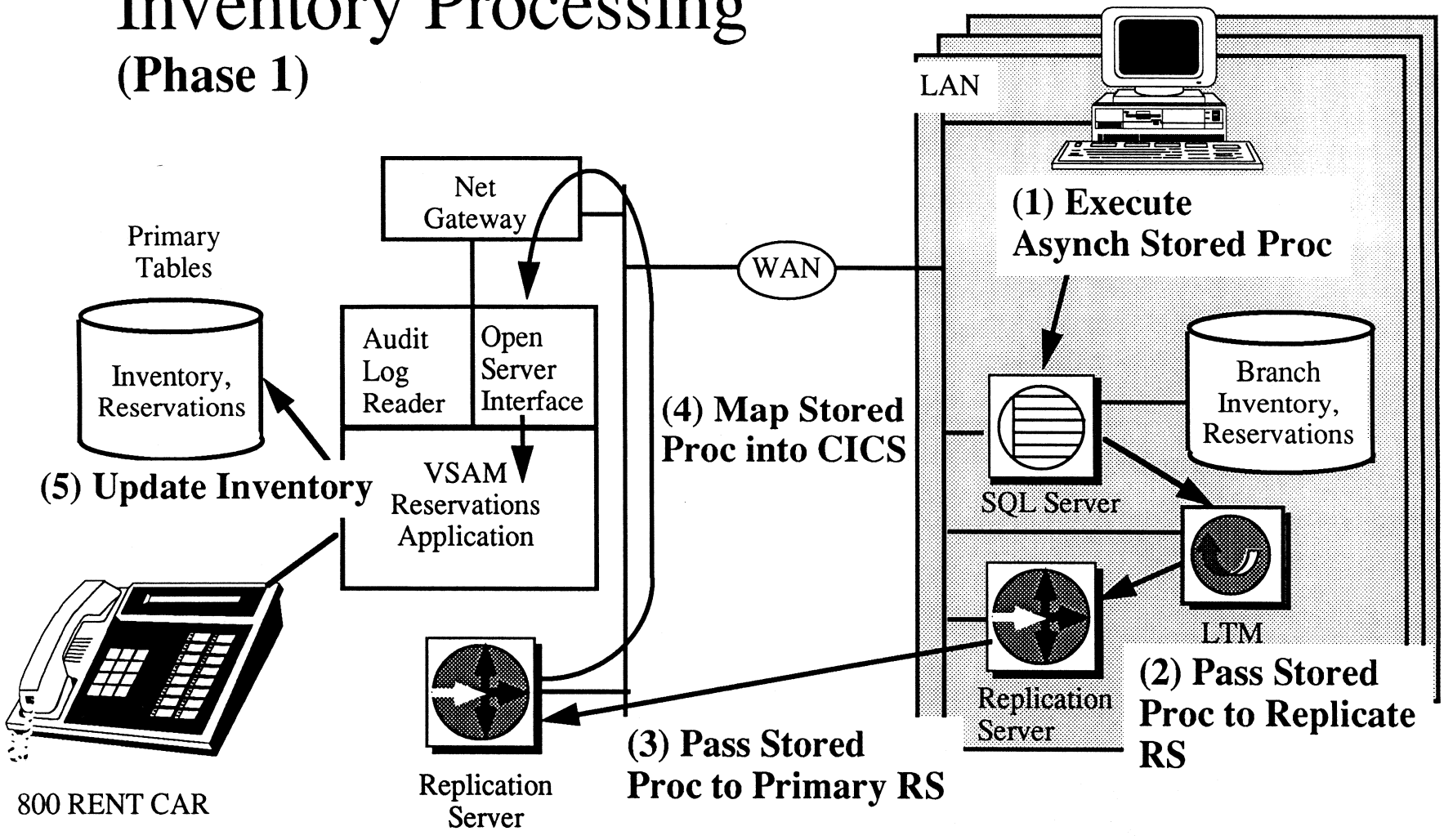
Replication Server Solution



Reservation Processing

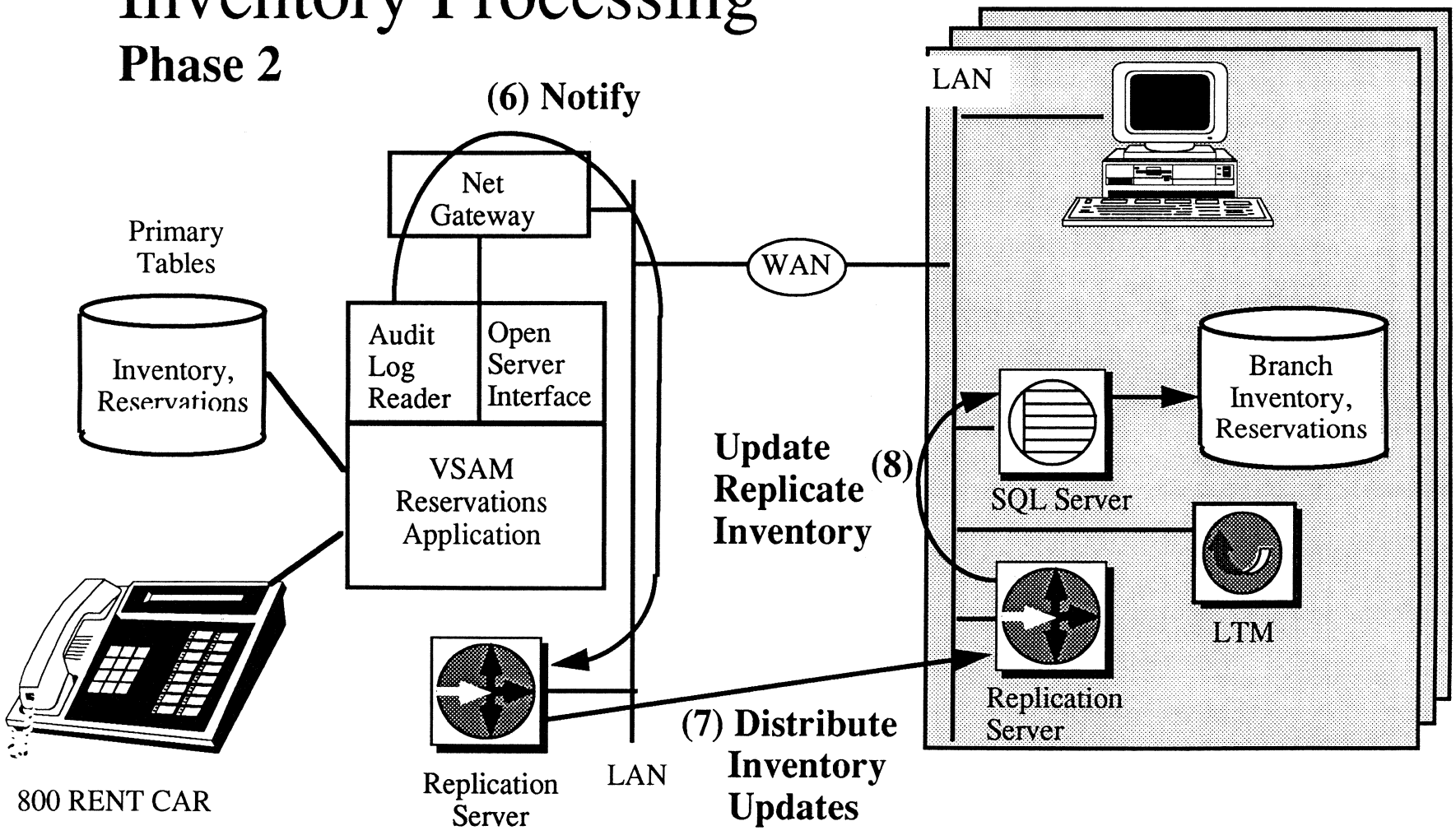


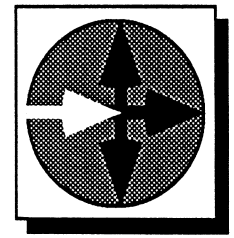
Inventory Processing
(Phase 1)



Inventory Processing

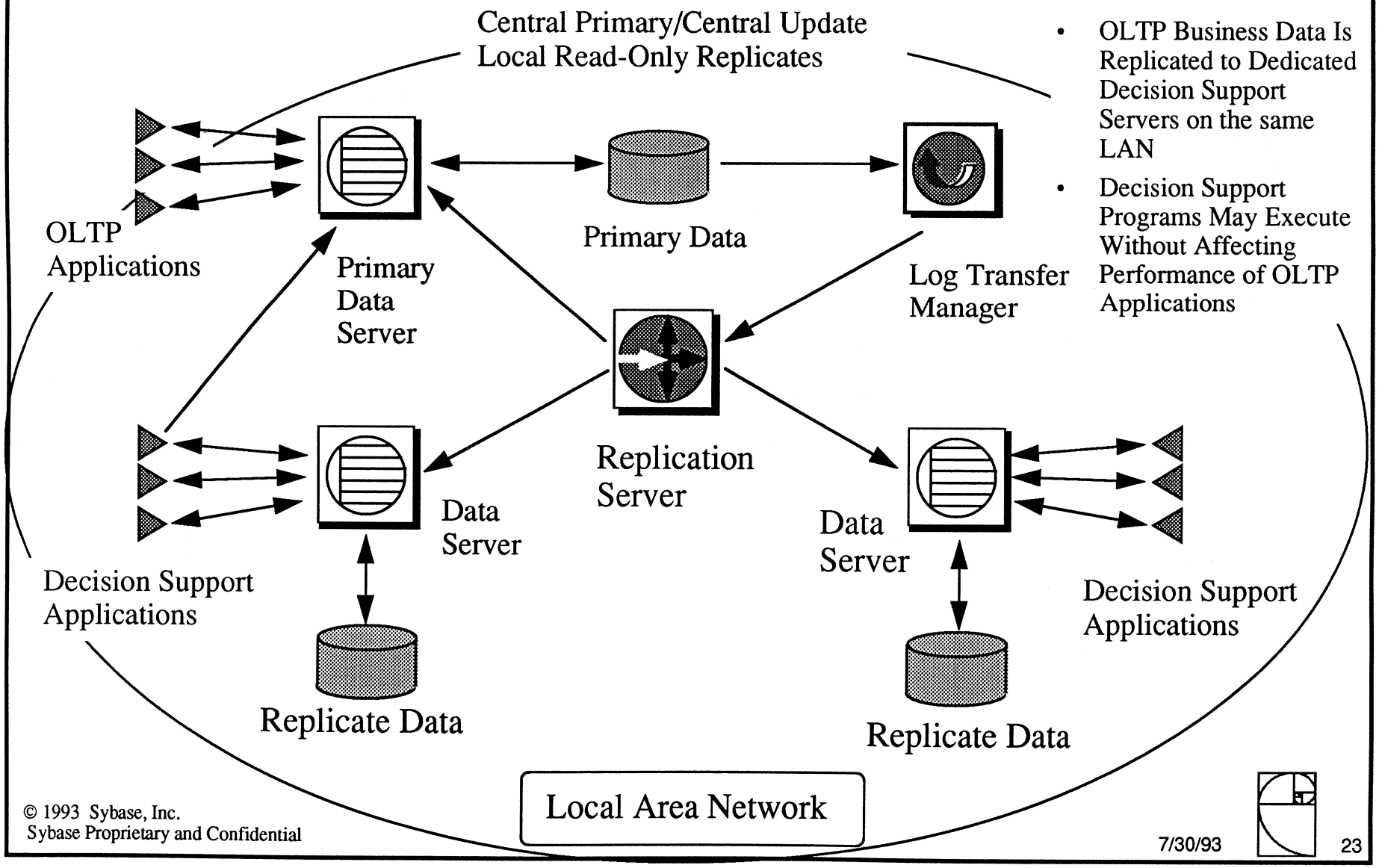
Phase 2





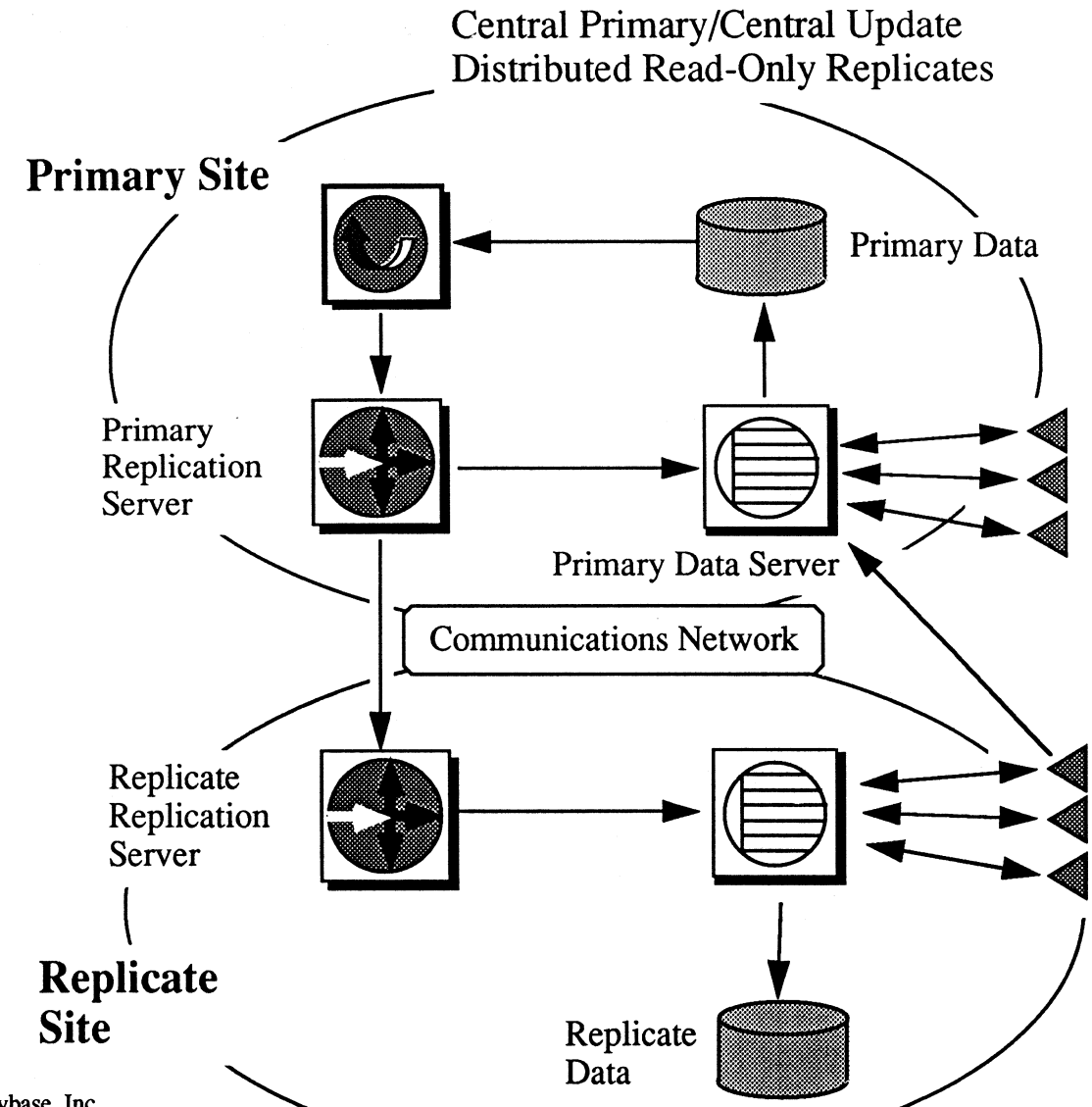
Replication Server Application Architectures

Replication Server Application Architectures



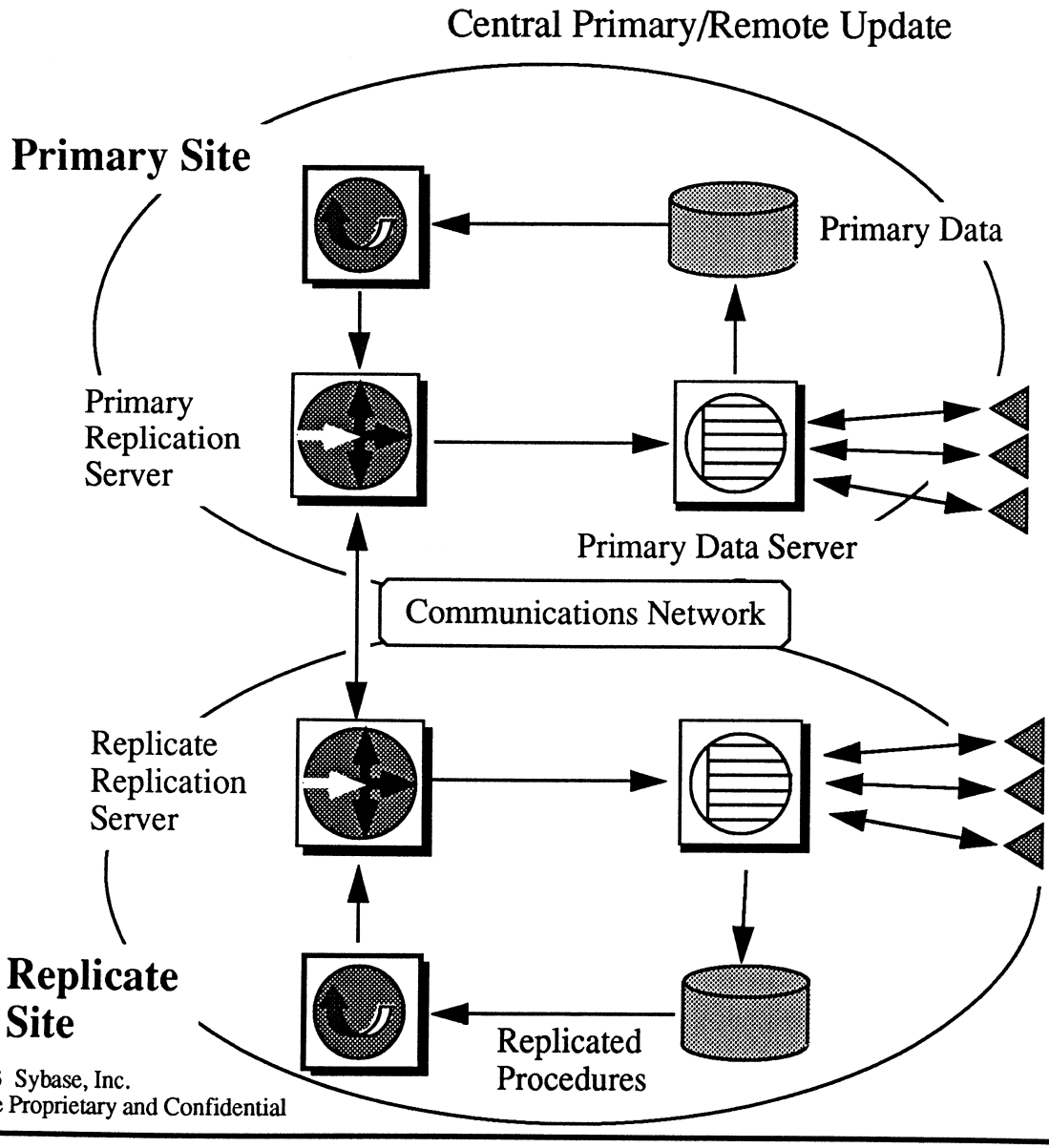
- OLTP Business Data Is Replicated to Dedicated Decision Support Servers on the same LAN
- Decision Support Programs May Execute Without Affecting Performance of OLTP Applications

Replication Server Application Architectures



- Remote Applications Use Replicate Data Read-Only, Update Primary Data Directly When Necessary
- Update Requests Must Be Deferred if Primary is not Available

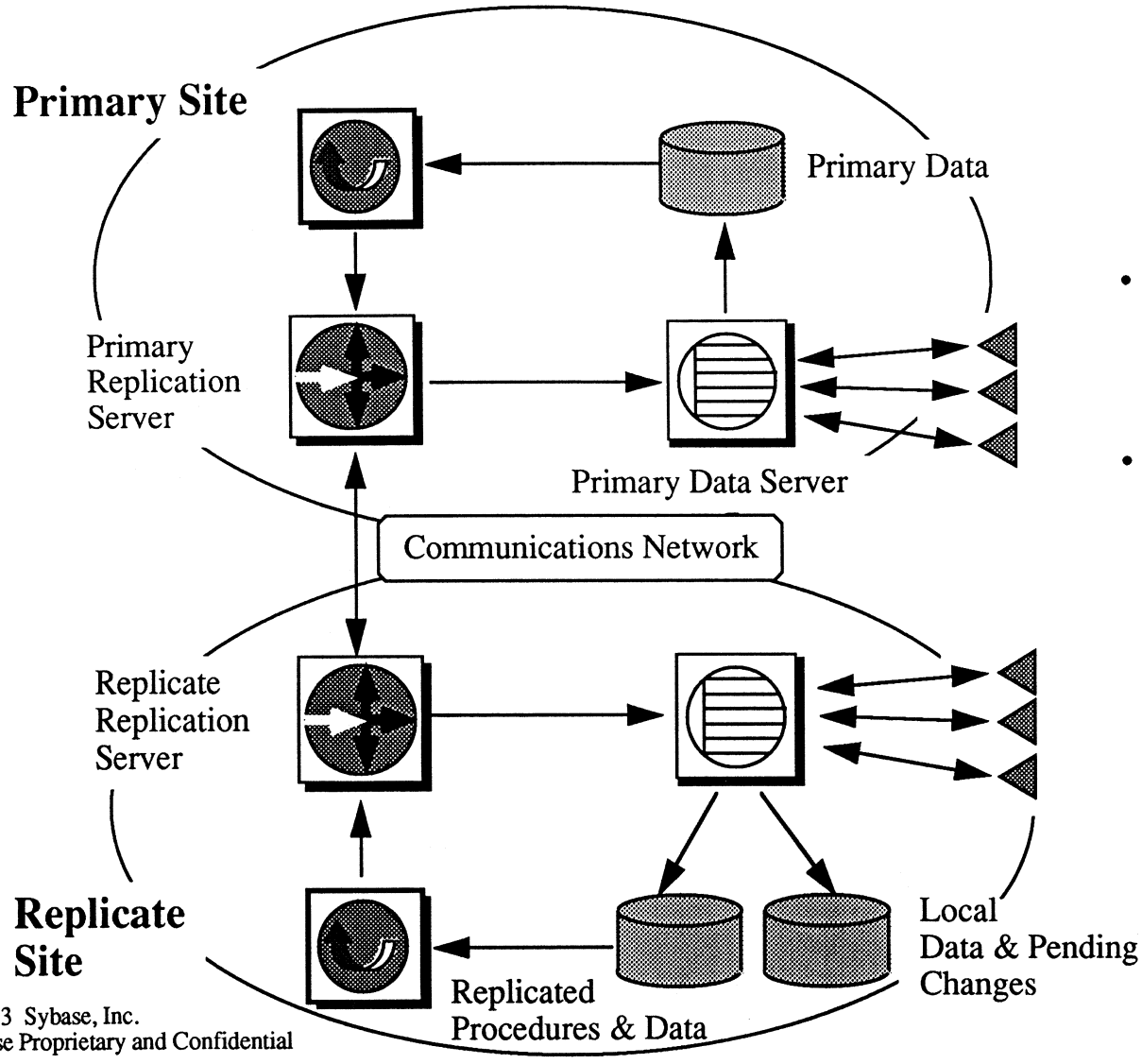
Replication Server Application Architectures



- Remote Applications use Asynchronous Stored Procedures to Update Replicate Data , Changes are not Visible to Remote Applications Until Primary Updates are Distributed.
- Remote Applications may Continue to Submit Updates when Primary is not Available

Replication Server Application Architectures

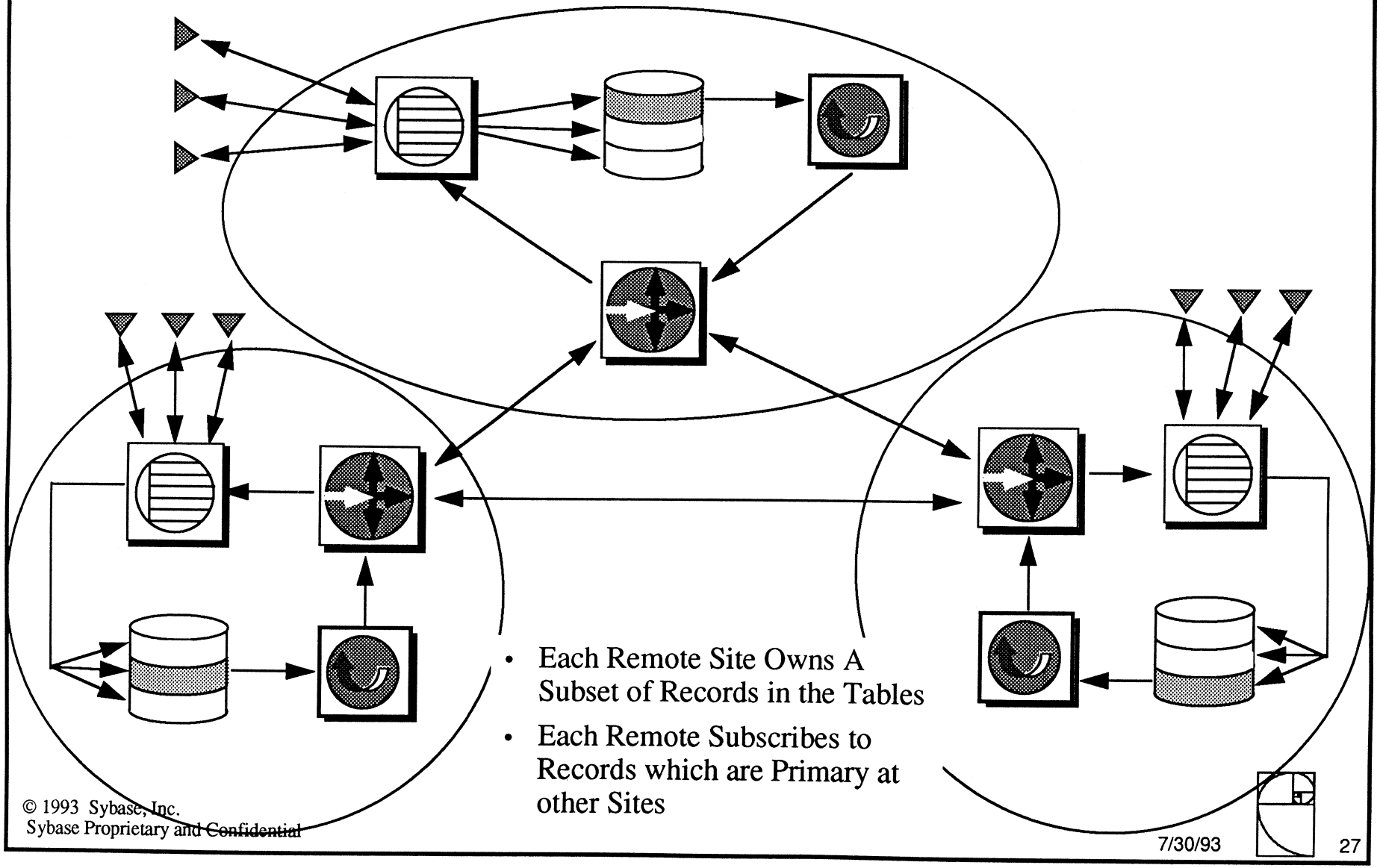
Central Primary/Remote Update/Local Changes



- Remote Applications use Asynchronous Stored Procedures to Update Replicate Data ,
- Pending Changes are Recorded Locally Until Primary Updates are Replicated, Therefore Visible Immediately

Replication Server Application Architectures

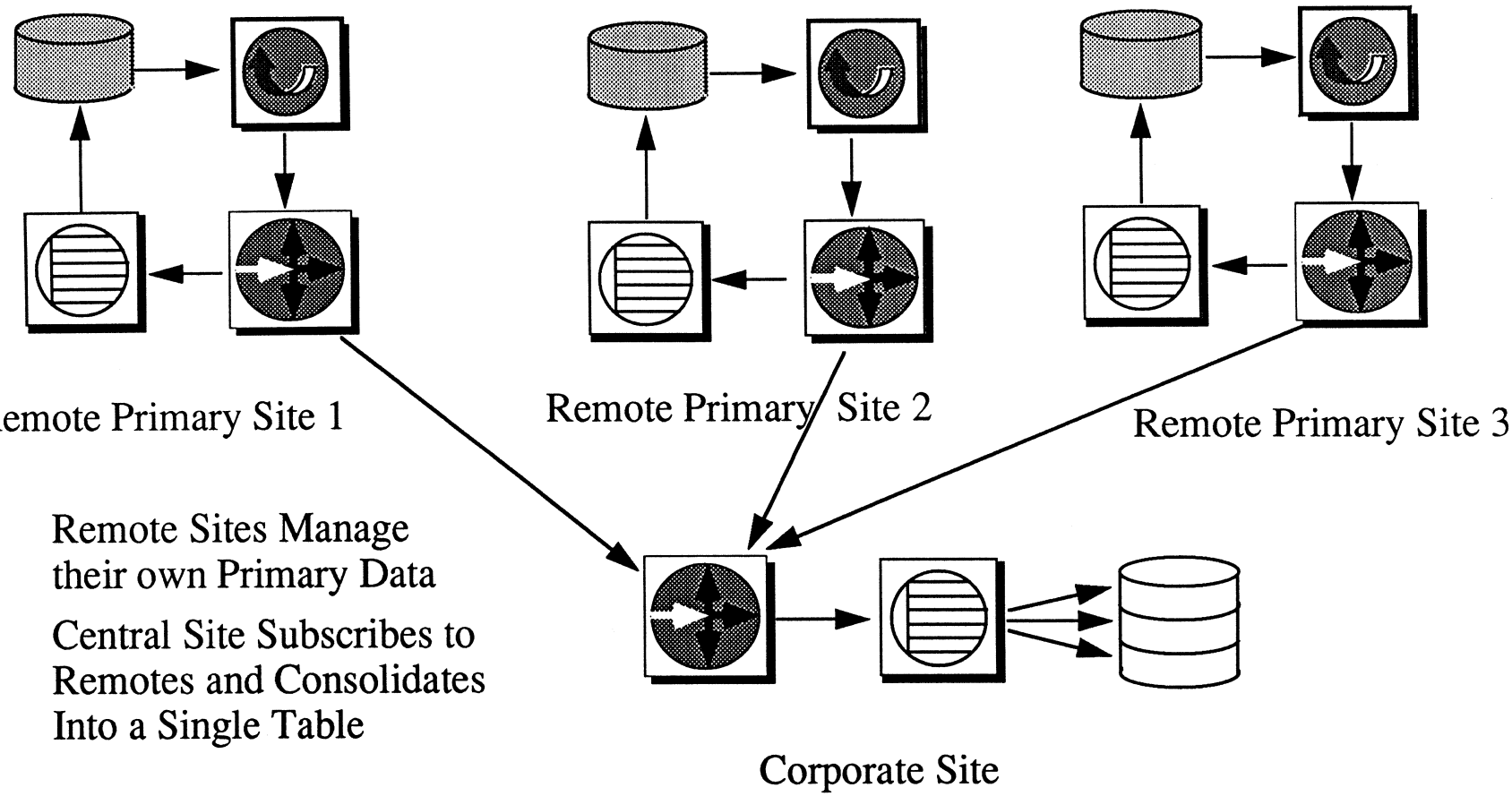
Distributed Partitioned Primary



- Each Remote Site Owns A Subset of Records in the Tables
- Each Remote Subscribes to Records which are Primary at other Sites

Replication Server Application Architectures

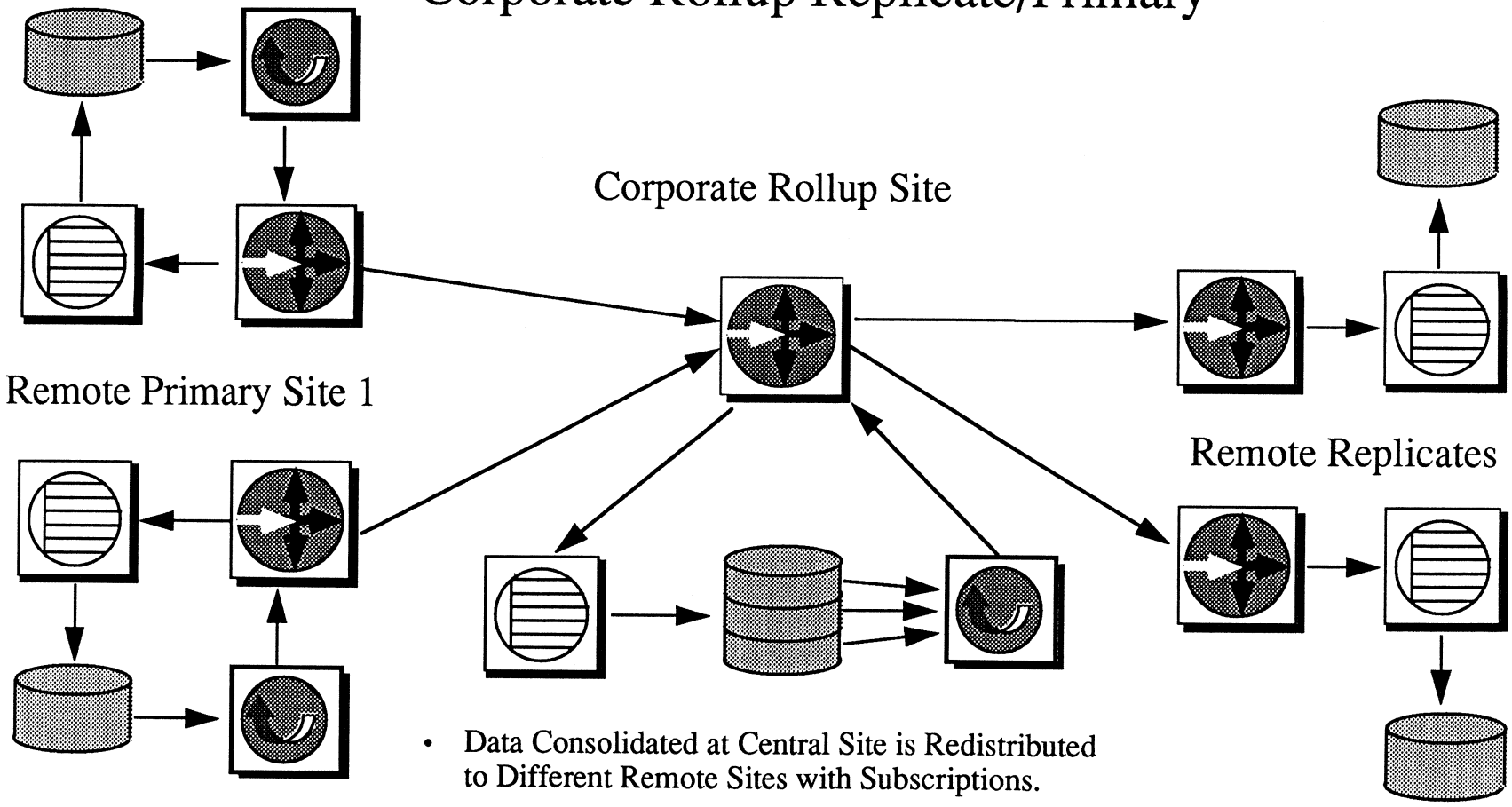
Corporate Rollup Replicate



- Remote Sites Manage their own Primary Data
- Central Site Subscribes to Remotes and Consolidates Into a Single Table

Replication Server Application Architectures

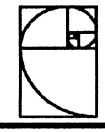
Corporate Rollup Replicate/Primary



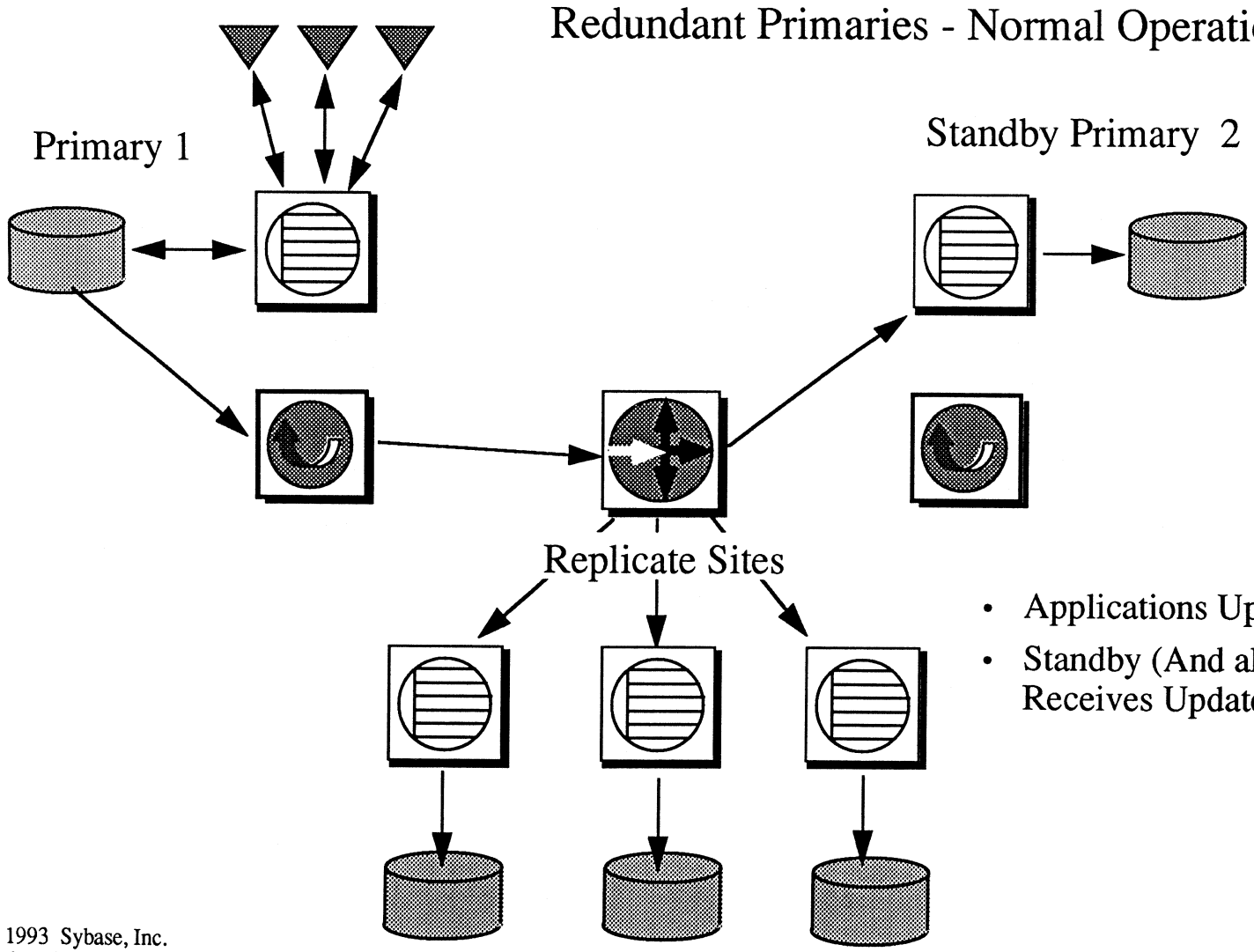
- Data Consolidated at Central Site is Redistributed to Different Remote Sites with Subscriptions.
- Requires two Replication Servers at the Central Site: one which manages the Rollup Table as a Replicate, the Other which Manages the Rollup Table as a Primary

Replication Server Application Architectures
Redundant Primaries

- Used As a Method for Warm Standby
- Also Used To Change Location of Primary At Designated Times
- Tables in Both Primary Databases Are Replicated
- Tables in Both Primary Databases Subscribe to Each Other
- Replicate Sites Must Subscribe to Both Primary Tables
- Applications Must use only one Database as the Primary at a Time
- Applications Must Reconcile Lost Transactions In Case of Primary Switchover



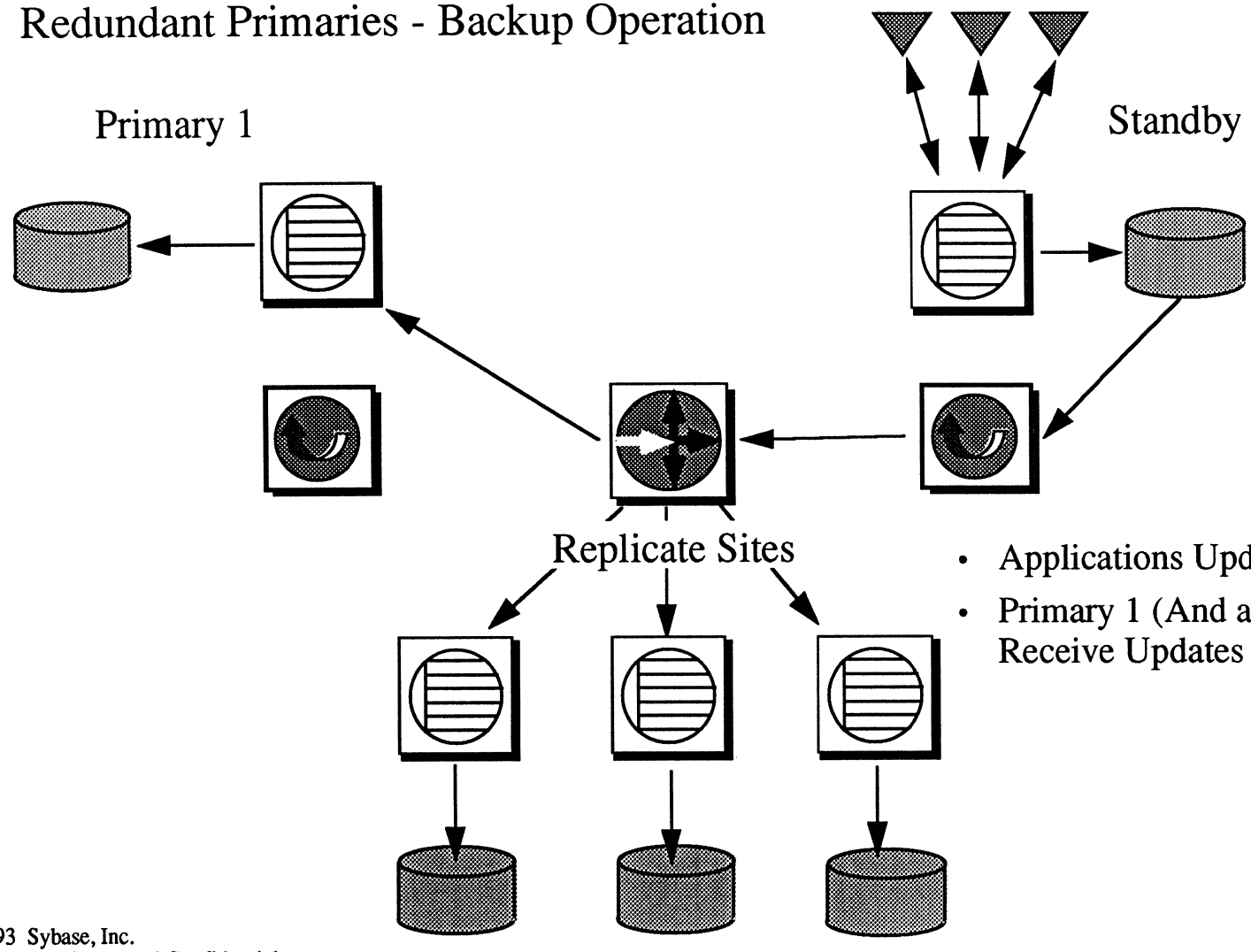
Replication Server Application Architectures



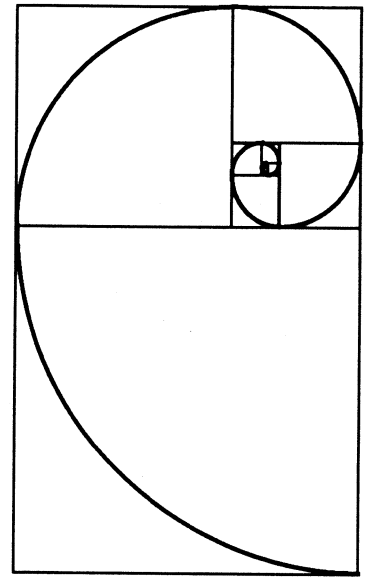
- Applications Update Primary 1
- Standby (And all other Replicates) Receives Updates from Primary 1

Replication Server Application Architectures

Redundant Primaries - Backup Operation



- Applications Update Primary 2
- Primary 1 (And all Replicates) Receive Updates from Primary 2

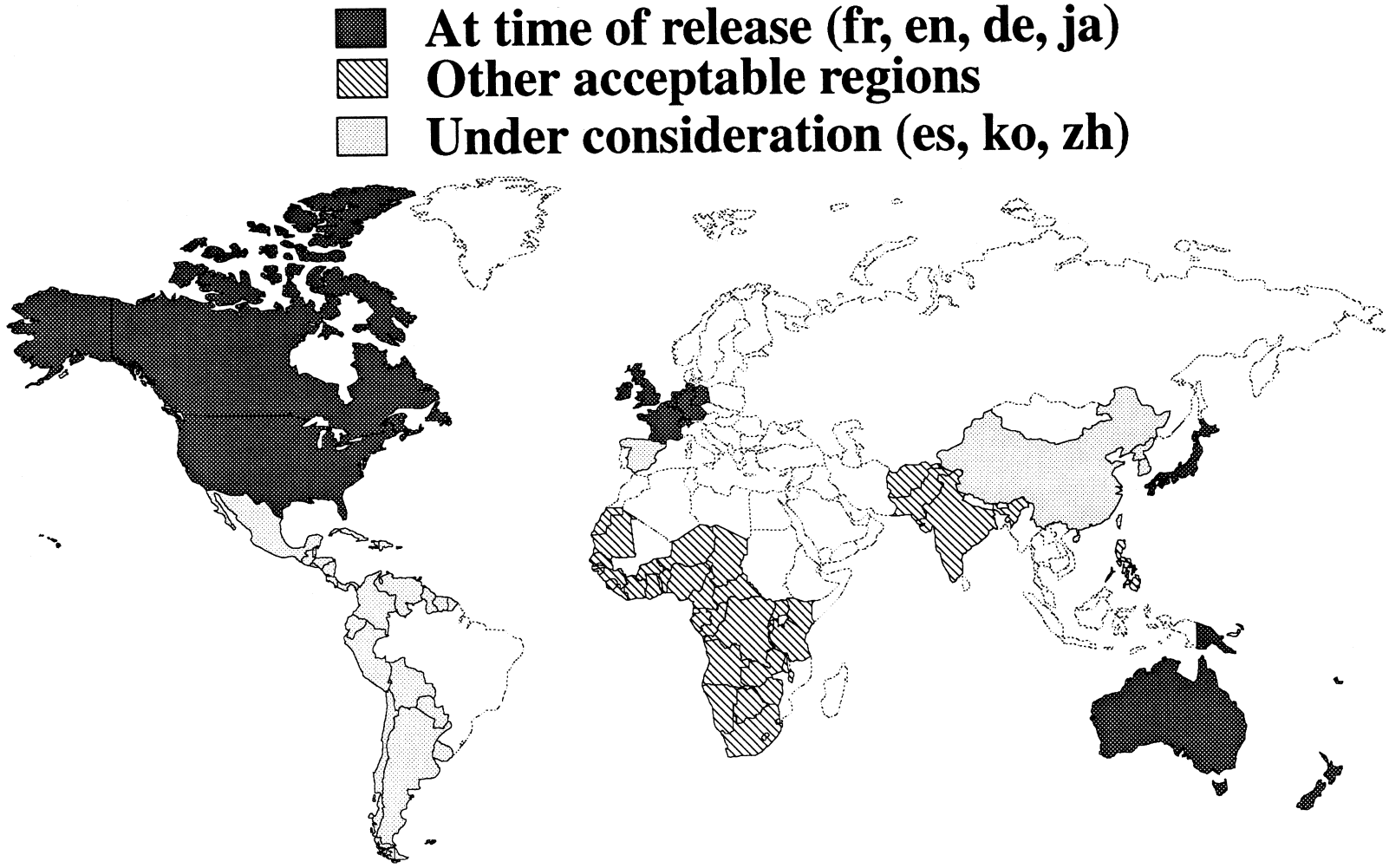


Writing International Applications with Sybase SQL Server and Open Client

By Mike McKenna

International & Secure SQL Server Development

Countries Supported by System 10



Example Installations

- English-only client-server environment
- Western, non-English, single-language environment
  - Homogeneous systems
  - Heterogeneous systems
- Western multi-lingual environment
  - Homogeneous systems
  - Heterogeneous systems
- Asian single-language environment
  - Homogeneous systems
  - Heterogeneous systems
- Special cases
  - Asian-to-western connectivity (incompatible character sets)
  - Non-supported languages
  - Cross-timezone installations

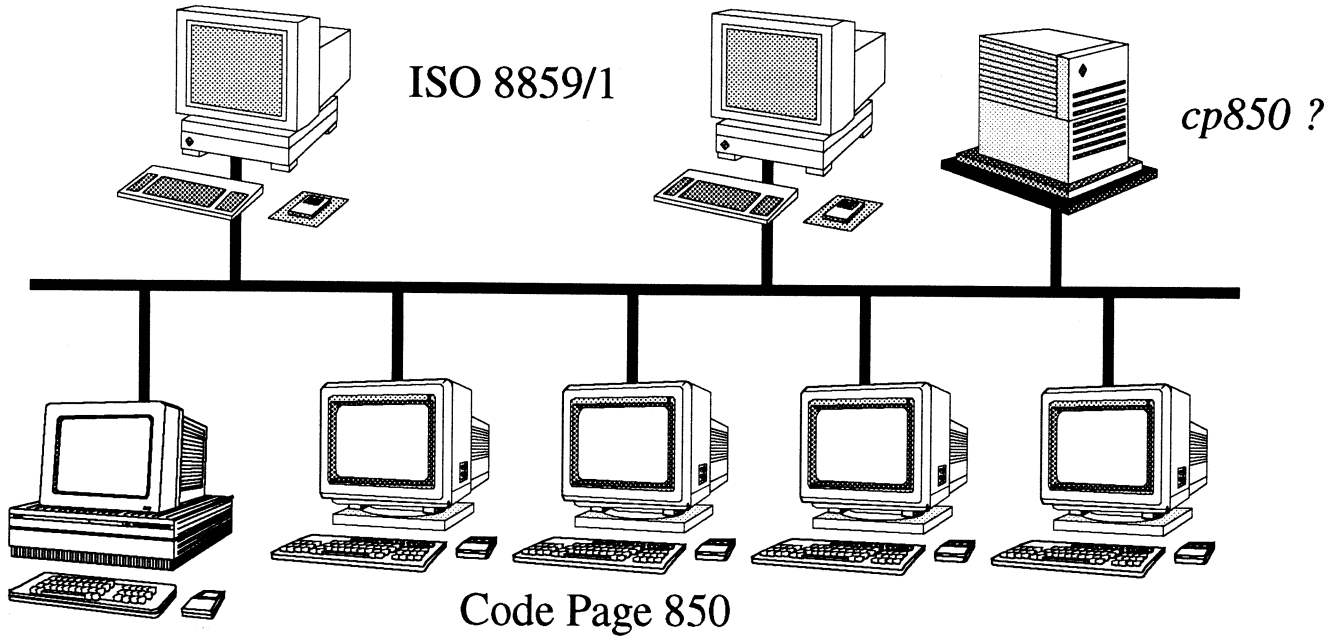
System 10 I18N SQL Server Features

- *(new for System 10.0)* Localized installation, ~~system boot, and error log~~
- Choice of character sets, languages, and sort orders
- Configurable date format
- *(new for Rel. 4.9)* Asian support for multibyte character sets
- *(new for Rel. 4.9)* Codeset conversion between a limited set of character sets
- *(new for Rel. 4.9)* User-accessible multilingual message handling via *sp_addmessage* and *sp_getmessage*
- *(new for Rel. 4.9)* Language-independent output string formatting using `PRINT` and `RAISERROR`

Systems Solutions

- Heterogeneous Environments
  - More than one character set throughout (e.g. - *iso_1, roman8, cp850)*

    At system design-time, choose default character set of majority of proposed client machines, and set up servers accordingly, or install additional character sets in servers and let server do codeset conversions.



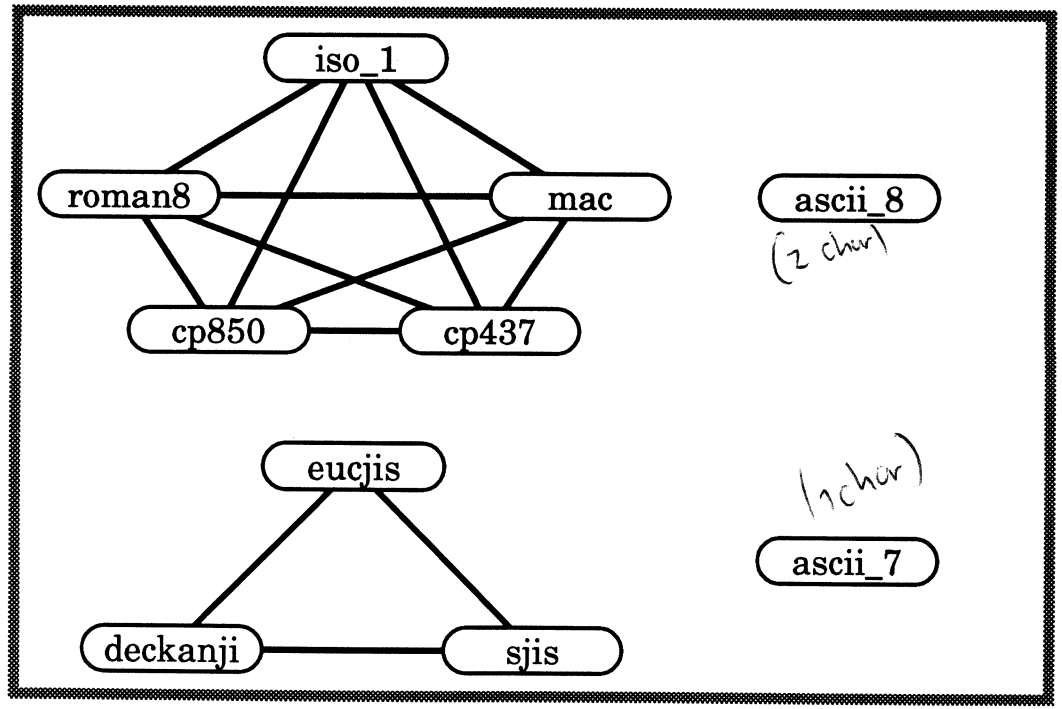
Character sets supported

Character Sets Supported

- Single-byte character sets:
  - ISO 8859-1 . . . . . "iso_1"
  - HP's Roman8 . . . . . "roman8"
  - Microsoft's CodePage 437 . . . . . "cp437"
  - Microsoft's CodePage 850 . . . . . "cp850" (PC)
  - Macintosh . . . . . "mac"
  - ASCII-7 . . . . . "ascii_7" → no translation to happen + you want to do it yourself
  - ASCII-8 . . . . . "ascii_8"

- Multi-byte character sets:
  - EUC-JIS . "eucjis"
  - Shift-JIS . "sjis"
  - DEC-Kanji"deckanji"

Codeset Conversions Supported



Sort Orders

The System 10 SQL Server supports one sort order per server. The four primary varieties are:

- Binary ordering . . . . . *binary.srt*
- Dictionary ordering . . . . . *dictionary.srt*
- Case-insensitive ordering . . . . . *nocase.srt*
- Case-insensitive with preference . . . . . *nocasepref.srt*

A few considerations when choosing the server default sort order:

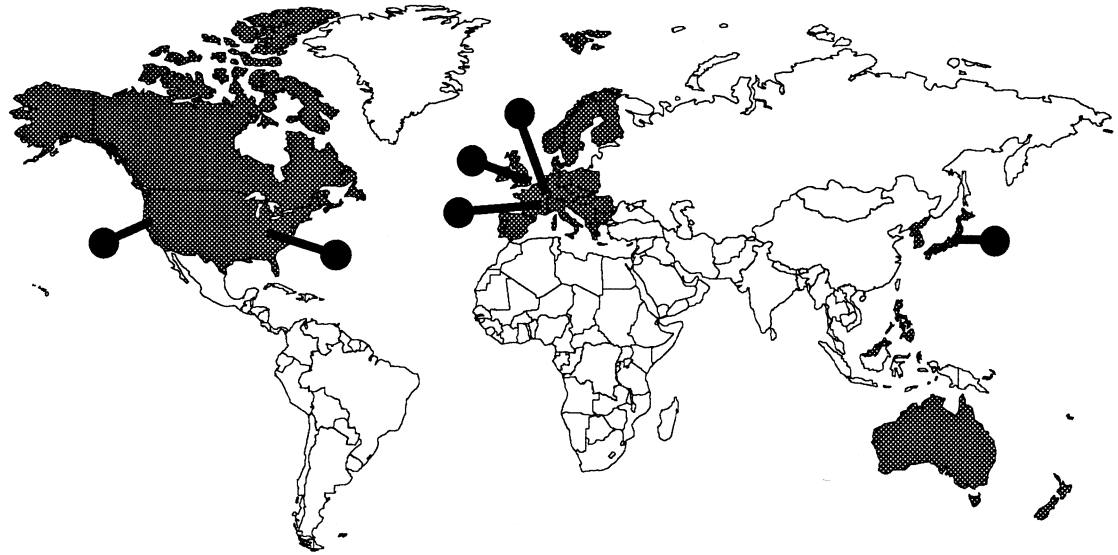
- ❑ Future distributed configurations
- ❑ End-user needs and client application abilities
- ❑ Performance
  - *binary.srt* is fastest
  - *nocasepref.srt* can incur up to 15% penalty for string-intensive processing

Sample Application

Global Manufacturing Operation

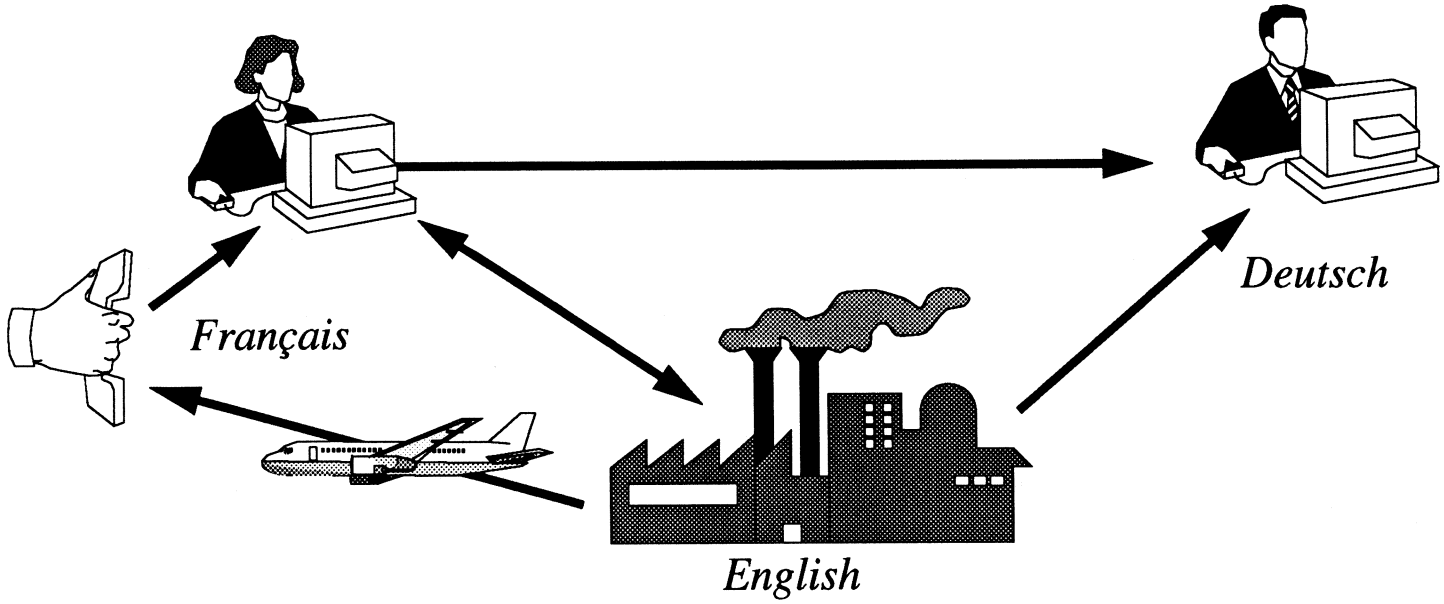
Suppose "Fübar, Gmbh", manufactures widgets. With headquarters in Munich, it has international offices in Tokyo, San Francisco, New York, London, Paris and Bonn. They need to sell to the EC, Pacific Rim, and North America.

They would like to be able to send out invoices, packing labels, and bills of lading in the destination language, and maintain reports in the language and cultural forms needed in the regional offices and the headquarters financial offices in München.



Application Requirements Example: Customer in Paris (French) orders 10,000 widgets.

- Data Entry Screens, customer invoice, bill of lading in *French*
- Inventory data indicate that the widgets must come from New York. Therefore, the shipping and inventory reports must be in *English*
- Financial reports must be in French, English and *German.*



Stored Business Rules

Stored Procedures, Triggers

For our example, we will assume the following rules:

- Any order over 5,000 widgets requires confirmation
- Based on inventory, the order-entry clerk is informed on stock available, and lead-time for delivery
- If stock goes below a certain set-point, an automatic order is executed to manufacture more widgets, based on economic rules set by corporate data
- If there are not enough widgets on hand world-wide to fill an order, an exception is raised to alert the order-entry clerk to inform the customer that the order will arrive in multiple pieces, the first batch from local stock, a second batch from regional warehouses, and finally, the final set when the next batch of widgets is shipped from a factory.

Stored Business Rules

Stored Procedures, Triggers (continued)

Example: Our French customer has made an order of 10,000 widgets. The table we set up requires and override field to be set by the client software before the insert is accepted. A message goes back to the Order-Entry clerk to reconfirm.

A sample trigger may look like:

```
create trigger Tr_BuyWidgets
on WidgetTable
for insert, update as
if exists (select * from inserted, ChkItem
        where inserted.NumUnits > ChkItem.Max
        and inserted.Override is NULL
        and inserted.Product = ChkItem.Product
        and ChkItem.Code = "confirm")
begin
        PRINT                   "Entry override needed"
        RAISERROR 30395         "Must confirm orders of more than 10,000 Widgets"
        ROLLBACK TRAN
        return (1)
end
```


Stored Business Rules (cont.)

The previous trigger can be "internationalized" by making it independent of natural language by using the features of *sp_getmessage,* `PRINT` and `RAISERROR`:

```
create trigger Tr_BuyWidgets
        .
        .
        .
begin
        declare @msg varchar(255)
        declare @maxunits int
        declare @product varchar(40)
        select @maxunits = ChkItem.Max, @product = ChkItem.ProdName
                from inserted, ChkItem
                where inserted.Product = ChkItem.Product
                        and ChkItem.Code = "confirm"
        /* MSG 20100: "Entry override needed" */
                sp_getmessage 20100, @msg output
                PRINT @msg
        /* MSG 30395: "Must confirm orders of more than %1! %2!."
                RAISERROR 30395, @maxunits, @product
                ROLLBACK TRAN
        return (1)
end
```


Print/Raiserror

`PRINT` and `RAISERROR` (as well as *dbstrbuild()* and *cs_strbuild()*) use the concept of a *format string* to allow position independent parameters. *%nn!* is used, where *nn* represents the position in the argument list.

For instance, the following is an English message:

- "%1! is not allowed in %2!", @arg1, @arg2

In German, it becomes:

- "%1! ist in %2! nicht zulässig.", @arg1, @arg2

In French, it is:

- "%1! n'est pas autorisé dans %2!.", @arg1, @arg2

And in Japanese, it's:

- "%2! の中で %1! は許されません。", @arg1, @arg2

Messages can be added in the home database of the applications intending to use them with *sp_addmessage*. The messages are retrieved based on the message ID and the value of *@@language*. This can be overridden with `SET LANGUAGE` or explicitly on invocation of *sp_getmessage*.

SQL Server Data Type Conversion

Date, Money

The SQL Server has localization capabilities for:

- Date/Time
  - Can choose order of year, month, and day for data entry and convert to CHAR by use of the *set dateformat* Transact SQL command. The valid settings are *mdy, dmy, ymd, ydm, myd,* or *dym.*
  - *convert(char, datetime_exp, style)* and *datepart()* functions will substitute the proper values for month and day names based on the default language. Can also use for data entry.
  - It is recommended, for portability reasons, to use the unambiguous ISO format of *yyyymmdd* for data entry from applications. This is guaranteed to work no matter what the *dateformat* setting is.
- Money
  - The SQL Server can only recognize prefix money delimiters ($, ¥, £) and does not prefix any symbols on conversion from money to character.
  - Recommend using client-side formatting.

Client Applications

Client/CS-Library and DB-Library I18N Features

Feature Set	*Client/CS-Library*	*DB-Library*
Setting/Getting Locale	ct_con_props / cs_locale	DBSETLCHARSET
		DBSETLNATLANG
		dbgetnatlang
		dbgetcharset
Codeset Conversion		dbcharsetconv
	use ct_command	dbservcharset
	cs_setconvert	dbxlate
String Comparisons	cs_locale	dbloadsort / dbfreesort
	cs_strcmp	dbstrcmp
		dbstrsort
String Formatting	cs_strbuild	dbstrbuild
	cs_convert	dbconvert
	cs_setconvert	
Date/Time	cs_dt_info	db{date,day,month}name
		dbdateorder

Connecting to SQL Server

Setting the Locale

Client/CS-Library

```
status = cs_loc_alloc(context,
        locale_ptr);
status = cs_locale(context,
        CS_SET, locale_ptr,
        CS_LC_ALL,
        LocaleName,
        CS_NULLTERM, NULL);
```

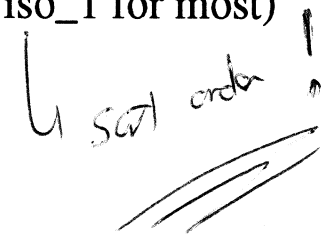
Looks up info in *$SYBASE/locales/locales.dat*. If LocaleName is NULL, will search environment for locale variables. Can individually set:

- CS_LC_ALL
- CS_LC_CTYPE
- CS_LC_MESSAGE
- CS_LC_TIME

DB-Library

```
DBSETLNATLANG(loginrec,
        LanguageName);
DBSETLCHARSET(loginrec,
        CharsetName);
```

DB-Library expects knowledge of LanguageName and CharsetName if different from the platform defaults (us_english, iso_1 for most)



Selecting the Language to Use

Determining the language to use for a given session can be done in several ways. Depending on the application, one way or another may be more desirable.

- SQL Server uses the following hierarchy to determine a session's language:
  1. If the T-SQL statement SET LANGUAGE "xyz" has been issued, then "xyz" will be the language in use for the session.
  2. Otherwise, if the client application specified a language in the login request packet — set via DBSETLNATLANG( ) or *cs_locale( ) / cs_config( )* or *ct_conn_props( )* — then this will be the language.
  3. Otherwise, if the user has a default language specified in syslogins.language, set via *sp_defaultlanguage*, then this will be the language in use.
  4. Otherwise, the language in use for the session will be the "default language" for the server, as configured in syscurconfigs.

Setting up languages - Example.

As an example, let's assume that an installation of SQL Server is primarily German based. We would set up the server's default language in this manner:

```
set language german
sp_configure "default language", @@langid
```

There is a user, however, named "andres" who typically wishes to use French. We set up the default language for "andres" in syslogins as follows:

```
sp_defaultlanguage "andres", "french"
```

Finally, the applications in use may be required to use the Posix locale of the user's current session. To ensure that whoever runs this application will be using the language of their default locale, the application specifies its language using Client-library's *cs_locale( )* function:

```
CS_LOCALE     *loc_ptr;
CS_CONTEXT    *context;
                :
                :
status = cs_locale(context, CS_SET, CS_LC_ALL, NULL, NULL, NULL);
                :
                :
```


How to Set Up Codeset Conversions

- Stand-alones (isql, bcp, defncopy) make use of the '-J' flag:

  No -J flag . . . . . *client's character set is platform dependent default*

  -J"charset_name" . . . . . *client's character set* is "charset_name"

  -J. . . . . *requests that **no** codeset conversions be done*

- DB-Library Applications make use of DBSETLCHARSET( ):

  No call to DBSETLCHARSET( ) . . *client's character set is platform dependent default*

  DBSETLCHARSET(login, charset_name) . . *client's character set* is "charset_name"

  DBSETLCHARSET(login, NULL) . . . . . *do **not** do codeset conversions*

- CT-Library Applications make use of cs_locale( ):

  No call to cs_locale( ). . . . . *client's character set is platform dependent default or inherits context*

  cs_locale(... , CS_LC_CTYPE, locale_name, ...) . . . . *client's character set is name from **$SYBASE/locales/locales.dat** in row associated with [platform]/language.charset*

  cs_con_props(connection, CS_SET, CS_CHARSETCNV, &CS_FALSE, sizeof(CS_FALSE), NULL)
  do **not** do codeset conversions

- T-SQL Interface makes use of the SET CHAR_CONVERT statement:

```
SET CHAR_CONVERT OFF
SET CHAR_CONVERT ON [ WITH { ERROR | NO_ERROR } ]
SET CHAR_CONVERT charset [ WITH { ERROR | NO_ERROR } ]
```


Client-Library Data Binding

cs_set_convert( )

A developer can use *cs_set_convert( )* to create custom datatype conversion routines. This powerful feature allows specific conversions to be set up at the context, connection, and data element levels.

cs_set_convert( ) can be used to convert data between:

- Standard Open Server and SQL Server datatypes
- Standard and user-defined types
- User-defined types

An example, based on the business model above, would be a currency conversion routine, where the SQL Server converts local currencies based on market info, but the client application can format it appropriately. First, define the datatypes in the SQL Server:

```
exec sp_addtype DM_money, money        /* Deutsche Marks */
exec sp_addtype FFR_money, money       /* French Francs */
exec sp_addtype USD_money, money       /* US Dollars */
```


cs_set_convert( ) (cont.)

Then, create the conversion routines:

```
CS_RETCODE
ConvMnyToChar(context, srcfmt, srcdata, destfmt, destdata, destlen)
CS_CONTEXT          *context;
CS_DATAFMT          *srcfmt;
CS_VOID             *srcdata;
CS_DATAFMT          *destfmt;
CS_VOID             *destdata;
CS_INT              *destlen;
{
        CS_RETCODE  status;
        CS_DATAFMT  dbl_fmt;
        double      mny_placeholder;
        CS_INT      dbl_len;
                    .
                    .
                    .
        status = cs_convert(context, srcfmt, srcdata, dbl_fmt,
                &mny_placeholder, *dbl_len);
        sprintf (destdata, "%s", pcurrency(mny_placeholder, srcfmt) );
        return (status)
}
```


cs_set_convert( ) (cont.)

Then create a conversion binding in the proper context. In this case, we would attach this binding for general use.

```
typedef    CS_MONEY              DM_MONEY;
typedef    DM_MONEY_TYPE         CS_USERTYPE + 1;

cs_set_convert(context, CS_SET, DM_MONEY_TYPE, CS_CHAR,
      ConvMnyToChar);
```

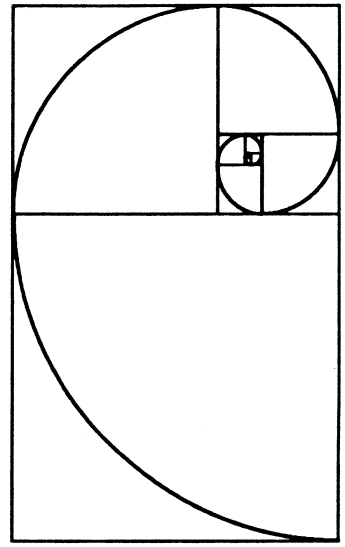
This same scheme can be used for:

- International Money
- Non-trivial date/time formatting
- Formatting of numeric values
- Western to Asian connectivity
- Non-supported codeset conversion

Summary

- The SQL Server and Open Client are fully internationalized.
- The SQL Server provides powerful features for multilingual heterogeneous solutions.
- The Open Client libraries provide very powerful and flexible solutions for global enterprise-wide connectivity solutions.
- Combined with evolving Posix standards, internationalization and localization of corporate systems can allow a developer to do a system design once locally, and then allow global deployment with little or no redesign later.

Steve Atherton



Sybase

Functional Training

Database Cursors

Author: Manish Chandra

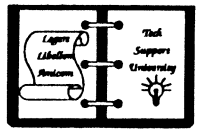
Database Cursors

- What is a Database Cursor?

  It is a mechanism for accessing the results of a SQL SELECT statement, one row at a time.

  Normally, a SELECT will return all the rows to the application in some order. Currently DB-Lib provides a mechanism to process each row. But the buffering is done purely by the client.

  With the System 10 SQL Server, the rows are returned to the application on demand rather than all at once.



Database Cursors

- What is a database cursor? (contd.)

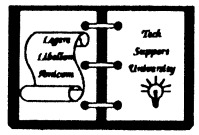
  There are two parts to a cursor definition:

  4. Cursor Result Set: The set of rows resulting from the execution of the SELECT.
  5. Cursor Position: A position in the result set. The cursor position reflects the current row.

Database Cursors

- Using the Cursor mechanism the applications can take action on each row rather than the entire set of rows returned by the SELECT.
- The applications can do Positioned Updates and Deletes.
- The cursor can be thought of as a handle on the result of a SELECT statement e.g

```
/* The following statement associates/declares a cursor on
** a SELECT statement.
*/
DECLARE test_curs CURSOR
FOR
SELECT name, title
FROM employees
```



Database Cursors

- Cursor name must be a valid SQL identifier.
- If the SELECT statement contains one of the following constructs, the cursor is defined to be read only:
  1. DISTINCT.
  2. GROUP BY
  3. Aggregates.
  4. UNION operator.
- Caution: Cursors using ORDER BY clause are updatable when declared in Ct-Lib or Precompilers but not when used in a stored procedure or trigger.

Database Cursors

- After a cursor has been attached to a SELECT statement, it has to be 'OPEN'. This means that the server should start logically processing this statement.

  OPEN test_curs

- At open time, they query is parsed and compiled but not executed.
- System 10 cursors optimize the processing at OPEN time. In most cases only partial processing is done at this time. This means that the entire result set is not generated at this point in time.
- There is no constraint on the maximum number of cursors which can be declared for a session.

@@SQLSTATUS

- A new global variable to hold status information resulting from the execution of a SQL statement
- Only the fetch statement will set it:

0	Successful completion of fetch
1	Fetch resulted in an error
2	There is no more data in the result set

Database Cursors

- The application can then access each row in the SELECT's result set by using the FETCH operation.

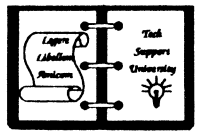
  FETCH test_curs

  This will return one row at a time.

- The results of a FETCH can be stored in variables/ parameters i.e

  FETCH test_curs INTO @name, @title

- The status of the FETCH is stored in a global variable called @@sqlstatus. By checking the value of this variable the applications can quickly tell whether the FETCH was successful or not.



Database Cursors

- If during processing the application wants to change a particular row, a positioned update can be issued. Let's say we want to change the title of a particular employee.

```
UPDATE employees
SET title = "Small Fry"
WHERE CURRENT OF test_curs
```

- The cursor scan position is not changed and the scan can continue from this point.

Database Cursors

- If the application wants to delete the current row then the following SQL can be used:

```
DELETE employees
WHERE CURRENT OF test_curs
```

- The cursor position is "before" the next row after a delete.
- You must do a fetch again, before you are allowed to update or delete the next row.

Database Cursors

- After processing as many rows as the application needs, the cursor can be closed. The closing of a cursor frees up valuable system resources and also resets the scan.

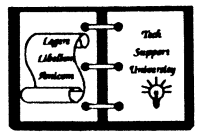
  It does not free all resources

- Before using the cursor again an OPEN must be issued.

  CLOSE test_curs

- System 10 SQL Server also provides another command DEALLOCATE to completely clean up the resources of a cursor. After issuing this command the cursor must be redeclared before it can be used.

  DEALLOCATE CURSOR test_curs



Database Cursors

- Cursors can be declared through the System 10 Precompilers and CT-library.
- Cursors can be declared in T-SQL, stored procedures and triggers.
- Cursors can be declared on the results of a stored procedure which contains a single SELECT statement. This can only be done through the pre-compiler or CT-library.

  Dynamic SQL uses this.
- To use cursors in DB-Library, you must submit them as T-SQL statements.

Database Cursors

- Cursors can remain open across transactions.
- A new SET option determines the effect of the transaction end on the open cursors.

  SET CLOSE ON ENDTRAN ON

- Leaving cursors open across transactions provides higher concurrency and performance.
- This SET option allows ANSI 89 compliance.

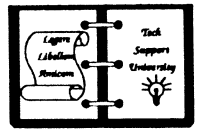
When using SET CLOSE ON ENDTRAN ON:

```
begin tran
    open cursor
commit tran (or rollback)
```

...will close the cursor.

```
open cursor
begin tran
commit tran
```

...will *not* close the cursor.



Database Cursors

EXAMPLE APPLICATION

- Design a stored procedure which does the monthly closing of bank accounts for The Small Community Bank.
- If the balance in the Checking account exceeds $1000 then an interest is added to the account.
- If the balance exceeds $10,000 then a bonus is also given to the account.
- However, if the balance is below $500 then a service charge is deducted from the account.
- The service charge, bonus and interest rates can all vary from month to month.

Database Cursors

- Let us first write the procedure without using database cursors.

```
/* The following procedure closes the accounts at the end of
** every month.
*/
CREATE PROCEDURE close_accts(
                    @int_rate NUMERIC(6, 6),
                    @serv_chg NUMERIC(8, 6),
                    @bonus NUMERIC(8, 6))
AS
/* Start a reconciling transaction */
BEGIN TRANSACTION
/* First update the accounts which are above $1000 */
UPDATE accounts
SET balance = balance + (@int_rate * balance)
WHERE balance >= $1000
```

Database Cursors

```
/* Next update the accounts which meet the bonus criterion */
UPDATE accounts
SET balance = balance + @bonus
WHERE balance >= $10000

/* Now deduct the service charge */
UPDATE accounts
SET balance = balance - @serv_chg
WHERE balance < $500

COMMIT TRANSACTION

RETURN
```

NOTE: This procedure scans the table three times.

Database Cursors

- Now let's rewrite the procedure using the database cursors.

```
/* The following procedure closes the accounts at the end of
** every month.
*/
CREATE PROCEDURE close_accts(
                    @int_rate NUMERIC(6, 6),
                    @serv_chg NUMERIC(8, 6),
                    @bonus NUMERIC(8, 6))
AS
DECLARE     @doupdate int
DECLARE     @balance   NUMERIC(20, 6)
/* Now declare a cursor on the SELECT from accounts */
DECLARE curs CURSOR
FOR
SELECT balance
FROM accounts
FOR UPDATE OF balance
```

Database Cursors

```
/* Next open the cursor and start the processing */
OPEN curs
/* fetch the first row */
FETCH curs INTO @balance
/* Now loop processing all the rows.
** @@sqlstatus = 1 - means error on previous fetch.
** @@sqlstatus = 2 - means end of result set reached.
*/
WHILE (@@sqlstatus!= 2)
BEGIN
    /* Always check for any errors first. */
    IF (@@sqlstatus = 1)
    BEGIN
            EXEC error_handler
            RETURN
    END
```

Database Cursors

```
/* Next compare the balance against different values */
IF (@balance < $500)
BEGIN
    SELECT @balance = @balance - @serv_chg
    SELECT @doupdate = 1
END
ELSE IF (@balance >= $1000)
BEGIN
    SELECT @balance = @balance * (1+@int_rate)
    IF (@balance >= $10000)
        SELECT @balance = @balance + @bonus
    SELECT @doupdate = 1
END
ELSE
BEGIN
    SELECT @doupdate = 0
END
```

Database Cursors

```
        IF (@doupdate = 1)
        BEGIN
            /* Next update the account with the new balance */
            UPDATE accounts
            SET balance = @balance
            WHERE CURRENT OF curs
        END

        FETCH curs INTO @balance
    END

    CLOSE curs
    RETURN
```

- NOTE: Each update commits as it is made.
- Exiting a stored procedure will close the cursor even if the "close" command is not issued.

Database Cursors

- We avoided multiple scans of the table.
- The second procedure used shorter transactions.
- Each row was touched once and kept locked for less time. This provides more concurrency and higher throughput.

Database Cursors

- Cursors can be declared to be read-only or updatable.

You should **always** specify, although is it optional.

This distinction helps the SQL Server in optimizing the queries.

If you are not going to update, you should explicitly use the read-only keyword.

It can be used to influence the locking behavior of the cursors.

The Update Lock is used to provide the stability needed for updatable cursors. Update locks have been used by the Server internally in previous releases. This is the first time it has been available externally.

It helps in making the application programs both more readable and maintainable.

If a cursor is declared for update, but some tables in the query will not be updated, use the *shared* keyword next to the table names.

The default lock mode, if not specified, is for update. However shared locks will be obtained for all selects. Exclusive locks will be obtained when the actual update is done.

```
DECLARE curs CURSOR
FOR
SELECT name, title
FROM employees
FOR UPDATE OF title
```

Database Cursors & the new locking mode

- Update Locking and Cursors:

  A table can be locked using Update locking mode when being accessed through a cursor.

  The update locks have the following properties:

  Update Lock conflicts with another Update Lock and Exclusive Lock.

  Update Lock is compatible with other Shared Locks.

- This means that after a Fetch, the scanner is assured that no other Fetch with an intention to update can be done on that page. This may result in fewer deadlocks. However, this may also reduce concurrency.

Database Cursors

- The following table shows the compatibility graph for the update locks.

Table 3: Lock Compatibility Graph

Lock Type	Shared	Update	Exclusive
Shared	Yes	Yes	No
Update	Yes	No	No
Exclusive	No	No	No

Database Cursors

- It is important to distinguish between various forms of cursors, as they differ slightly in their resource usage characteristics:

  1. **Client Cursor:** The cursor is declared through the Open Client Library (or Embedded SQL which in turn generated Ct-Lib calls). The Ct-Library keeps track of these cursors also and buffers them for the application.
  2. **Execute Cursor:** This is a subset of the Client Cursors and refers to the set of cursors defined on the result of a stored procedure. The procedure is associated with the cursor at OPEN time and can be reused by another cursor after this cursor id CLOSEd.

Database Cursors

3. **Server Cursors:** These are cursors declared in stored procedures. The client is not aware of these cursors. The client sees the result of a FETCH as a normal SQL SELECT statement.

   When the deallocate statement is issued, memory is not released, but the name is available.

4. **Language cursors:** These cursors are declared in pure SQL and resemble Server Cursors in their behavior.

   Deallocate *does* release memory with these cursors as well as make the name available. The declare must be the only statement in the batch.

Database Cursors

- Cursors and row positions

  The SQL Server dynamically maintains tables and indexes. This means that during a cursor scan, other updates by the same connection to these tables can interfere with the scan.

  The SQL Server uses unique indexes for scanning the tables to avoid these problems. An example of such a problem is a table with no clustered index where if a row changes size it migrates to the end of the table.

  In any case, the columns which make up the index being used to access the tables should never be changed or else the rows can reappear or disappear.

Database Cursors

- The following sequence of events illustrates what happens when a row is changed (known as the Halloween problem):
  1. Cursor 1 points to Page 1, the 2nd row on that page.
  2. Page 1 is completely full.
  3. The row is modified in the same connection so that it's size increases.
  4. The row is moved to a new page. If there is no clustered index the row is moved to the last page of the table.
  5. The cursor position is changed internally by the SQL server to point to the next row on the page of the table.
  6. This row will reappear in the scan later.

Database Cursors

- If however the scan had been on a unique index or the table had a unique clustered index then:
  1. Instead of the row moving to the end of the table it'll expand in relatively the same position.
  2. The SQL Server will internally move the cursor position to this new physical position.
  3. As a result the scan remains consistent.
- Cursors require a unique index for updates

  If your cursor is declared for update, you will immediately get an error that no unique index exists if it is missing.

  If your cursor is not declared for update and no unique index exists, your selects will be fine. As soon as you try to do an update the cursor will be closed. You will not get an error until you try to use the cursor again.

Database Cursors

- Cursors and Identity

  The new Identity property columns can be used to influence the cursor's selection of certain access paths for tables.

  Any index which includes the identity property column is implicitly available for cursor's access paths.

  The main advantage of not marking an index unique is to avoid deferred updates when they are not needed.

Database Cursors

- Exercise: Create a trigger on a salary table which checks to see:
  1. Whether the new salary level is less than the salary level of the employee's supervisor.
  2. It should also check that the new salary level is not more than the maximum salary level specified for that department.
  3. Hint: Use cursors on inserted and/or deleted tables.

Database Cursors

- The schema for the example is shown below:

  CREATE TABLE depts(deptid NUMERIC(5, 0) PRIMARY KEY, deptname CHAR(40), max_salary NUMERIC(15, 4))

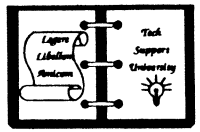
  CREATE TABLE employees(empid NUMERIC(10, 0) IDENTITY PRIMARY KEY, manager NUMERIC(10, 0), empname CHAR(30), deptid NUMERIC(5, 0))

  CREATE TABLE salary(empid NUMERIC(10, 0) UNIQUE, salary_level NUMERIC(15, 4))

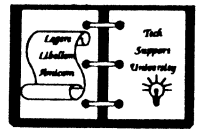
Database Cursors

- The following trigger uses cursors on inserted and deleted tables to accomplish the rules.

```
CREATE TRIGGER sal_constr
ON salary
FOR INSERT, UPDATE
AS
DECLARE @empid NUMERIC(10, 0),
        @salary_level NUMERIC(15, 4),
        @manager NUMERIC(10, 0),
        @manager_salary NUMERIC(15, 4),
        @dept_maxsql NUMERIC(15, 4),
        @deptid NUMERIC(5, 0)

DECLARE ins_curs CURSOR
FOR
SELECT empid, salary_level
FROM inserted
```




```
OPEN ins_curs
FETCH ins_curs INTO @empid, @salary_level

WHILE (@@sqlstatus!= 2)
BEGIN
    /* Get the manager and department of the employee */
    SELECT @manager=manager, @deptid=deptid
    FROM employees
    WHERE empid = @empid

    /* Get the manager's salary */
    SELECT @manager_salary = salary_level
    FROM salary
    WHERE empid = @manager

    /* Get the maximum salary for the department */
    SELECT @dept_maxsal = max_salary
    FROM depts
    WHERE deptid = @deptid
```




```
        /* Business Rule 1 */
        IF (@salary_level > @manager_salary)
            ROLLBACK TRANSACTION

        /* Business Rule 2 */
        IF (@salary_level > @dept_maxsal)
            ROLLBACK TRANSACTION

        FETCH ins_curs
END

CLOSE ins_curs
RETURN
```

Database Cursors

- Cursors and permission checking: All permissions on the base tables for SELECT permissions is done at OPEN time.
- This means that if the permission is revoked after the cursor has been OPEN the subsequent FETCHes will succeed.
- However, when a cursor is closed and then re-opened later the permission check is done again.

Database Cursors

- The number of rows fetched to the client can be configured by a SET command per cursor:

  SET CURSOR ROWS 20 FOR test_curs

- Each FETCH will return 20 rows. However Ct-Lib provides the option to buffer these rows at the client end.
- Typically client applications would use this option to minimize network traffic.
- Caution: If this number is set too high then cursors degenerate to the old Db-Lib functionality of dbgetnextrow(), which means if you try to update the current row, it will be the 20th row

  This functionality is not availabe in ISQL.

Database Cursors

- Global variable @@rowcount is set to the total number of rows returned so far for the cursor.
- This is slightly un-intuitive but makes sense once you consider the fact that the number of rows returned on each FETCH is set by the application.

Database Cursors

- A design for applications using cursors should consider the following factors:

4. How long will my cursor remain open?
5. How many cursors will be open simultaneously?

   Answers to 1 & 2 will tell you the resource usage characteristics of your application.

6. Will I be doing updates to the scanned tables in the same process when the cursor is open?

   If this is the case, then the cursor scan will be affected in much the same way as when these updates were done through the cursor.

7. What is my criterion for uniquely identifying rows in the tables?

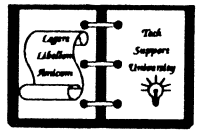
   The answer here could be a primary key which does not change or a column with the new Identity property.

8. Is the cursor really necessary here?

   Once cursors are available, they can be abused. Remember the Relational Model is essentially a Set Oriented Model.

Database Cursors

- Cursors are not needed when:
  1. The rows to be selected have the same uniform qualification criteria.
  2. The same operation is done to every row.
- Example: Give a bonus to all accounts which have a balance more than $2,000. Here a single update is much better than using a cursor to decide which rows to update.



Database Cursors

- The example about reconciling of accounts also contained an interesting problem. The use of @do_update variable could have been avoided, had I decided to use the following SELECT which restricted the rows sent by the server.

```
SELECT balance
FROM accounts
FOR UPDATE OF balance
WHERE balance BETWEEN 500 AND 1000
```

Tools Applications

Developers

Presentations

SYBASE
Gain*Momentum* 2.0

Market and Product Overview

July 1993

By Mitch Bishop

The Enterprise Client/Server Company™

Client/server computing has begun to play a prominent role in the information strategies of many organizations. Many companies have already reaped the enormous benefits of client/server applications at the departmental level and are starting to expand their client/server focus outward to the enterprise at large. As these companies move from legacy systems to client/server architecture, they face sizable application development challenges.

Perhaps the biggest challenge is to deliver new applications that are flexible enough to respond to the demands of today's dynamic global business environment. These applications must be deployed rapidly and cost effectively, and they must meet the diverse information needs of all end users – both within the organization and outside. At the same time, new applications must help companies achieve key business requirements: a time to market that is as short as possible, customer satisfaction, product quality, and profitability. These factors require new and innovative approaches to application design and development.

This paper discusses the trend toward client/server computing and discusses the approaches that today's corporate developers are taking toward client/server application development. It then describes Gain*Momentum*® 2.0 – Sybase's powerful, object-oriented application development environment. Gain*Momentum* 2.0 is an extensible, integrated framework for building and distributing information-rich client/server applications for UNIX and Windows NT platforms. With Gain*Momentum* 2.0, Sybase provides a bridge between mainstream, forms-based applications and the next generation: scalable client/server applications built from reusable multimedia data objects.

From Back Office to Extended Office

As it has from the beginning, the computer industry continues to expand its focus outward – giving ever-broader classes of users richer and more powerful ways to leverage information.

The introduction of mainframe technology more than 30 years ago focused on automating *back office* business functions – activities related to data capture, storage, and retrieval, which required significant amounts of repetitive clerical effort. Early computer systems automated these manual tasks with monolithic applications that displayed monochrome alphanumeric characters on nonprogrammable terminals connected to a host processor. Such generic applications required dedicated, trained operators and were consequently limited to the back office domain.

The advent of the minicomputer and the later proliferation of personal computers drove the demand for *front office* applications. Departments and individuals dedicated to accomplishing specialized, business-specific tasks, such as accounting, engineering, manufacturing, marketing, or human resources use front office applications. These departmental users are more knowledgeable in the broader business processes of their organizations and require more flexible access to data at the point of entry or inquiry. Front office applications typically incorporate a forms-based graphical-user interface (GUI) to simplify common functions and interactions and to reduce the need for user training and support.

The Client/Server Revolution

With the advent of its SYBASE® family of relational database products in 1987,

Sybase, Inc., introduced the concept of client/server architecture – the separation of applications into distributed components across a network. Client/server computing has emerged as the dominant computing paradigm of the 1990s, and companies in extendedly every industry have successfully deployed strategic client/server business applications. As these companies achieve success with client/server computing at the departmental level, they are now adopting strategies to link departmental applications into the mainstream, MIS computing environment. Now the client/server market is evolving rapidly toward its next phase: Enterprise Client/Server℠, characterized by mission-critical applications that integrate and share information across the entire enterprise.

An Outward Expansion

As the compelling economic and competitive benefits of enterprise client/server computing begin to pay off, companies will step up the expansion of client/server applications outside the boundaries of the office. These *extended office* applications facilitate global business transactions with suppliers, distributors, partners, and customers through real-time exchange of information for strategic decision-making and business planning. Extended office applications make it possible to share information and computing power across the entire value chain of an organization: from suppliers who automatically receive messages relevant to inventory levels, to strategic partners who share in the development of new products or markets, to distributors who require on-line access to sales and marketing data, and ultimately to customers who can access services interactively from remote locations.

Because of the diversity of the user base, extended office applications demand a highly flexible user interface that is driven by user intuition and completely eliminates the need for training. More specifically, the interface might need to include several languages and might incorporate media such as animated graphics that use both audio and video. In addition, the interface must allow for easy, transparent navigation through vast data sources on remote servers.

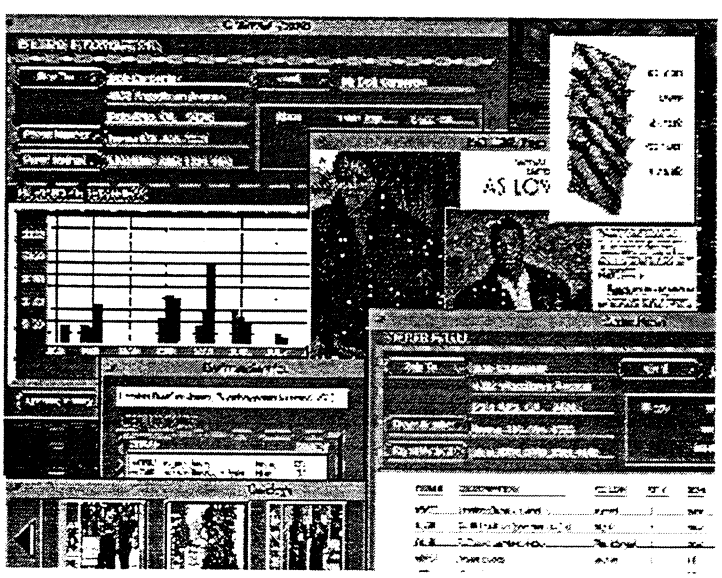


Figure 1. Gain*Momentum* 2.0 applications incorporate the look and feel of a business to help users make decisions based on more complete understanding of information.

Addressing the Productivity Gap

A major factor driving the evolution of computing is the explosion of processing power and memory on the desktop. Features such as high-resolution graphic displays, built-in CD-quality audio, full-motion video decompression, and real-time access to large corporate databases are widespread and will soon be standard for desktop computers. As such, these computers can now deliver the power necessary for a new generation of front office and extended office client/server applications.

Paradoxically, as computing power has increased, productivity gains for both developers and end users are disappointingly low. Application development time is still measured in years and the application backlog continues to grow. An enormous opportunity – and a formidable challenge – for client/server computing is to reverse this decline and leverage the capabilities of today's powerful desktop platforms to increase productivity throughout the enterprise. Increasing productivity requires successfully moving legacy back office applications to more cost-effective and productive front office and extended office client/server applications.

Gain*Momentum* 2.0 provides a comprehensive application development architecture for rapid migration of legacy applications to forms-based GUI applications. At the same time, Gain*Momentum* 2.0 supplies ample headroom to extend these applications to meet future requirements. For example, a near-term strategy for moving a legacy inventory system to a client/server architecture may be to use a forms-based GUI initially. Later, the application can be extended to include capabilities for letting users interact with visualizations of products or access video clips that provide comprehensive product demos. By clicking a mouse or touching a screen, a strategic planner might browse through a combination of SQL-based product specifications and multimedia-based information about product use to improve business plans and decisions. Applications that combine powerful client/server access with a multimedia user interface allow users to draw on information presented in various ways from widely disparate sources and make conclusions based on much more complete information than was previously available. The ultimate result is higher-quality output and increased productivity.

The Gain*Momentum* 2.0 Solution

Sybase Gain*Momentum* 2.0 is an object-oriented software architecture for rapid development and deployment of client/server applications. Developers can use Gain*Momentum* 2.0 to prototype, develop, and test powerful applications that model the look and feel of the business, delivering the ease of use, interoperability, and relational database access required for maximum productivity in today's strategic information environments.

Gain*Momentum* 2.0 provides an extensible, integrated toolset for building open, portable applications that may stand alone or interoperate with SQL database servers. Within a single software environment, Gain*Momentum* 2.0 developers can create, edit, and link all forms of multimedia objects; access and manipulate SQL data from SYBASE or ORACLE servers; script highly sophisticated application interactions; and deploy application runtimes across heterogeneous client/server platforms.

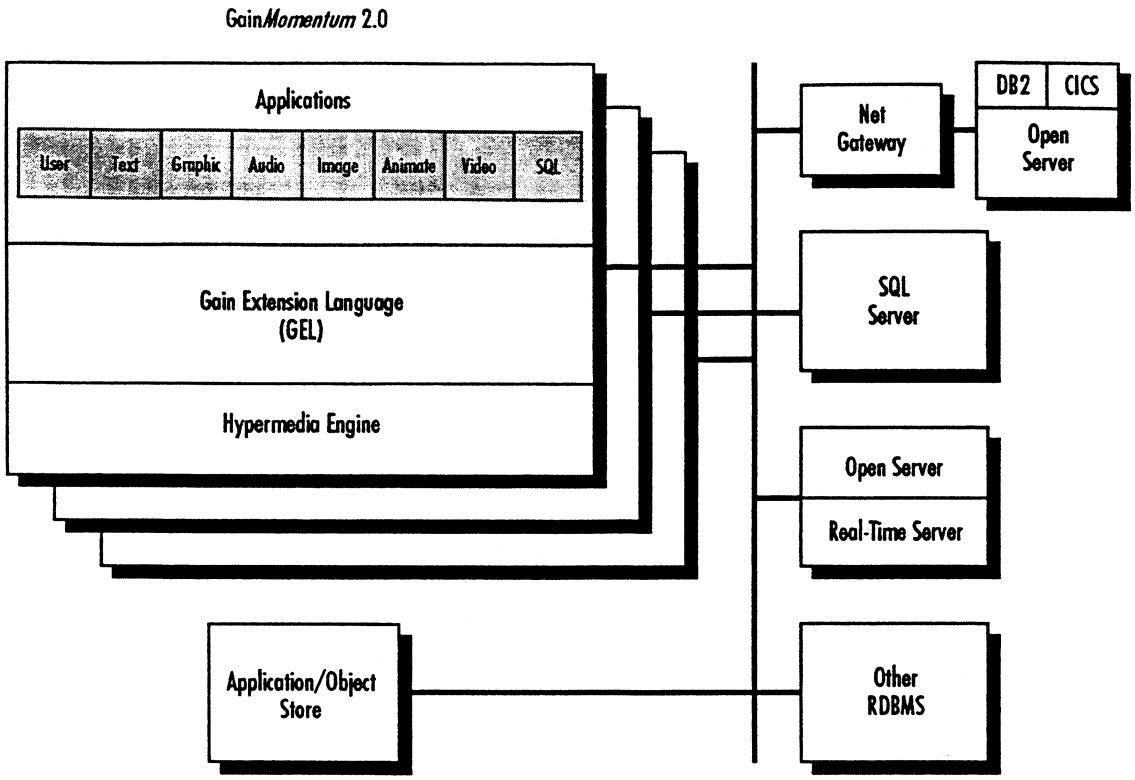


Figure 2. Gain*Momentum* 2.0 is complete application development architecture, providing and facilities for building, implementing and managing large-scale client/server applications.

Gain*Momentum* 2.0 supplies two unique architectural features. First is the Gain Extension Language (GEL), an object-oriented, multimedia 4GL-based scripting language from which Gain*Momentum* 2.0 itself was developed. Second is an advanced Application/Object Store, which enables the effective management of complex objects and their properties. These objects include encapsulated application libraries; user interface control objects, such as buttons, menu bars, and lists; and time-based multimedia objects, such as audio, video, and animated graphics.

Gain*Momentum* 2.0 also furnishes seamless, code-free access to RDBMS data, a full suite of visual tools for manipulating a wide variety of multimedia datatypes, and powerful functions for managing and administering of multiuser application projects. These include multilevel access permissions, database check-in/check-out, and version control.

Currently in worldwide distribution, Gain*Momentum* 2.0 supports the development of highly interactive applications ranging from electronic catalogs and point-of-transaction kiosks to employee performance support systems (EPSS) that integrate on-line help, reference, and training; RDBMS systems for managing financial services transactions; military command and control; and decision support.

The Future Is Built In

Gain*Momentum* 2.0 bridges the gap between mainstream, forms-based applications and next-generation scalable, client/server applications composed of reusable multimedia objects. Its extensible architecture permits modular application design and gives companies the flexibility to build the future into their long-term application development strategies. At present, many developers still use 3GL procedural interfaces. For them, even forms-based GUIs represent a great change. The architecture of Gain*Momentum* 2.0 takes into account the enormously varied state of the software development industry in creating a product that supports the needs of current practices, yet is sufficiently advanced that developers can add new modules to an already existing foundation, rather than starting over from scratch, as new requirements emerge. When COBOL developers want to add multimedia to their applications, the tools technology will already be there. By building the future into today's applications, companies can avoid the high price of discarding and replacing obsolete applications every few years.

Developing Forms-Based, Transaction-Oriented Applications

Using Gain*Momentum* 2.0's SQL Data Manager tools, developers can rapidly prototype, test, and implement forms-based applications running against any SQL RDBMS. This is accomplished through an extensible point-and-click interface that frees developers to concentrate on application functionality rather than writing SQL code. Although a wide variety of conventional 3GL- and 4GL-based tools are available for building forms-based applications, all but Gain*Momentum* 2.0 are narrow "point solutions" for building front ends to RDBMS data. Gain*Momentum* 2.0 fully supports RDBMS data but goes far beyond these point solutions by providing:

- A complete suite of integrated multimedia editors for comprehensive control of all data types
- An extensible, object-oriented 4GL
- The Application/Object Store, which manages all elements of the application regardless of format or source

Developing Object-Oriented, Multimedia-Based, Event-Driven Applications

The Gain*Momentum* 2.0 architecture takes full advantage of object technology, multimedia, and event-driven programming to deliver highly interactive and intuitive applications for even the most nontechnical end users. GainMomentum 2.0 helps deliver such applications through two key architectural features:

- The Application/Object Store is a powerful, built-in set of services for organizing, storing, and sharing high-bandwidth information types, such as complex images, audio, and video. It gives developers complete control over all data objects in an application. These objects include those resident at the client, such as GUI definitions, scripts for handling user- or data-driven events, and logic for processing SQL queries. The Application/Object Store also manages multiuser development functions such as object check-in/check-out, version control, and multilevel access permissions.
- Gain Extension Language (GEL) is Gain*Momentum* 2.0's multimedia 4GL and programmatic framework. With more than 1600 primitives and functions, it offers developers unlimited programming flexibility so they can expand and extend the Gain*Momentum* 2.0 tool environment itself. Using GEL, developers can build "tools within tools" to accommodate even the most site-specific application requirements.

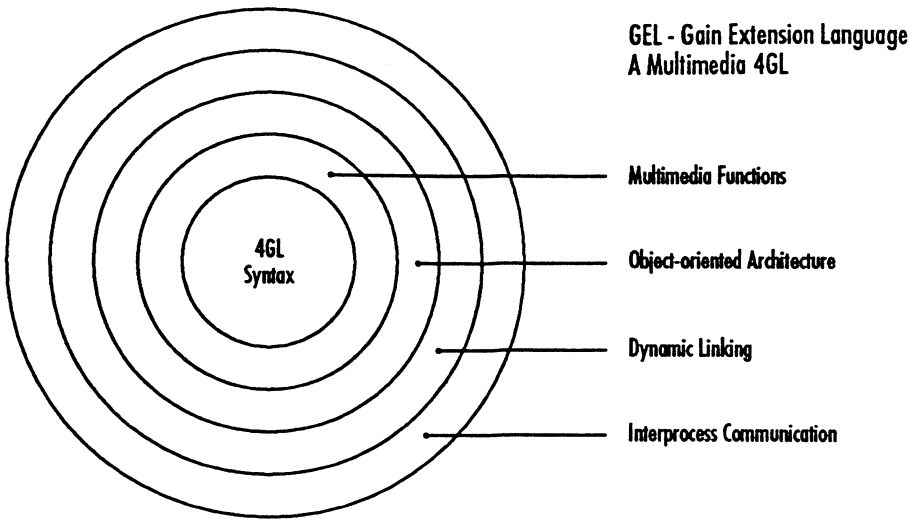


Figure 3. The Gain Extension Language goes far beyond traditional 4GL languages to support the next generation of scalable client/server applications built from reusable multimedia objects.

Productivity for Developers and End Users

Many factors drive the economics of computing. Computing organizations often emphasize high budgets for hardware procurement and software development, failing to recognize the high costs associated with application implementation and ongoing use. Indeed, initial development costs for a large system may be high. But they pale when compared to the costs of maintaining that system over time, providing ongoing system training and support, and paying the pro-rated salaries of the hundreds or thousands of people using that system.

Although most application development tools focus on adding value only for the development phase, Gain*Momentum* 2.0 supports the entire application life cycle.

Design and Development

Gain*Momentum* 2.0 decreases development time by providing tools that are easy to learn and use. Developers then can use Gain*Momentum* 2.0's GEL 4GL to build custom tools that substantially reduce development effort by automating repetitive, application-specific functions. All data in Gain*Momentum* 2.0 applications are managed as objects, with properties, methods, and behavior readily accessible from pop-up property sheets – so, data can be easily shared and reused. User feedback is often critical to successful development, and Gain*Momentum* 2.0's rapid design and prototyping capabilities allow developers to take advantage of that feedback from an early stage. This can save an enormous amount of time because developers will no longer have to rewrite application behavior that has proven impractical under actual user conditions.

Implementation

Implementing applications with Gain*Momentum* 2.0 requires substantially fewer resources than with traditional tools because application training, help, and support can be integrated directly into the application. Employing Gain*Momentum* 2.0's CAI Option and Gain*Exposure*™

Option, users can easily learn even the most complex applications through on-line computer-based training, context-sensitive help, and self-running simulations. In addition, Gain*Momentum* 2.0 can be linked to FrameMaker and Interleaf documents. Application development with Gain*Momentum* 2.0 requires a single software investment, unlike other tools that cannot support training, help, reference systems, and other important application functions.

Use and Maintenance

Gain*Momentum* 2.0 applications deliver substantial economic benefit during ongoing use and maintenance. Because Gain*Momentum* 2.0 supports the integration of high-bandwidth multimedia datatypes, application ease of use can be dramatically improved. A picture may be worth a thousand words, but the combined power of sound, color, and action opens up entirely new dimensions in application productivity. Visualization of information enhances comprehension, and motivation and stimulates creativity, – whether the application has been designed to facilitate processing an insurance claim or selecting a 401(k) pension plan. The productivity benefits of modeling the user's real-world environment and evoking a familiar and secure context, are easily measured in increased quantity and quality of output for any given application system.

Gain*Momentum* 2.0 applications help IS organizations overcome the pitfalls of application maintenance. The structure of a Gain*Momentum* 2.0 application is based on objects that can easily be modified or replaced to keep up with changing application requirements. To make a change, developers can simply modify the properties of an existing application object or replace it with another object without affecting the rest of the application.

Gain*Momentum* 2.0 System Architecture

The Gain*Momentum* 2.0 system offers a powerful environment for building client/server applications that blend seamlessly with an organization's existing corporate data and applications. Its software supports heterogeneous network configurations based on open systems hardware platforms and prevailing connectivity standards, which shield developers and end users from the complexity of underlying hardware and network issues.

Open Standards Support for Integration of External Data

Integrating existing SQL data from various sources, Gain*Momentum* 2.0 applications can connect directly to SYBASE and ORACLE servers and retrieve data from other relational databases (such as DB2 and Rdb) through a Sybase OmniSQL Gateway™. Legacy data stored on IBM mainframes in VSAM or other nonrelational formats can be accessed from Gain*Momentum* 2.0 using the Sybase Open Server™ for CICS.

Additionally, external data in the form of formatted text, bitmap images, vector graphics, and audio can be readily incorporated into Gain*Momentum* 2.0 applications. External applications can interoperate with these applications using interprocess communications standards such as Internet sockets, ToolTalk, or UNIX pipes.

An Environment Optimized for Network Delivery

Gain*Momentum* 2.0 supports the full spectrum of network solutions, ranging from small PC-based homogeneous LANs sharing data only (available early 1994) to large-scale WANs operating across ISDN and distributing data and applications to thousands of users in diverse locations.

To minimize network bandwidth requirements, the data in Gain*Momentum* 2.0 applications is stored and distributed over the network in an efficient compressed format. Users can further tailor applications to the bandwidth parameters of specific media by modifying the application design to match underlying network considerations.

Applications are stored in Gain*Momentum* 2.0's integrated Application/Object Store, which can be distributed across the network for local retrieval of data at the client machine. Additionally, applications can be tuned using Gain*Momentum* 2.0's unique predictive fetching algorithms to optimize delivery of high-bandwidth data over the network.

Complete Application Portability

Once created, Gain*Momentum* 2.0 applications are portable to any of the supported hardware platforms presented in Table 1. Platform dependencies are already resolved within the Gain*Momentum* 2.0 engine, so developers need not concern themselves with the logistics of recompiling and optimization. All user interface widgets are interchangeable among Motif, Windows (available early 1994), and OPEN LOOK, ensuring a consistent look and feel on the target deployment platform. For example, a Gain*Momentum* 2.0 application developed on a Sun SPARC workstation with a Motif GUI could be deployed on a Windows NT or Windows 32S client without modification by the developer.

Visual Development Environment

Gain*Momentum* 2.0's visual development environment uses a suite of point-and-click media editors to facilitate development of an application's visual content as well as a substantial portion of an application's behavior. Using these editors, developers can perform many diverse of development tasks without programming. These tasks include laying out the screens in an application; creating or importing text, graphics, and audio objects; incorporating GUI objects such as buttons, lists and sliders; linking objects together; and attaching actions to objects to respond to user-driven or data-driven events.

Platform	Operating System
SunSPARC	SunOS Solaris
IBM RS/6000	AIX
HP 9000/700	HP UX
SGI Indigo	IRIX*
DEC Alpha	OSF/1* Open VMS*
Intel X86	Windows NT* Windows 3.1*

*In Development

Table 1. Gain*Momentum* 2.0 applications are portable across a wide range of workstation platforms. Applications built on one platform may be presented on any other without recompiling or code modifications.

An Extensible Multimedia 4GL

Developers use Gain*Momentum* 2.0's feature-rich visual media editors to create an application's look and feel, import and edit external data, design and manipulate multimedia objects, and implement basic application behavior. Special or additional application functionality occurs though scripting in the Gain Extension Language (GEL).

An interpreted 4GL, GEL allows for rapid design iterations, development, and testing. GEL extends the traditional 4GL paradigm by providing primitives and functions for multimedia data manipulation, object orientation, custom tools, and interprocess communication. In deployment, Gain*Momentum* 2.0 applications use a run-time byte-code interpreter with a preliminary compilation step to improve performance.

Application/Object Store

At the very heart of the Gain*Momentum* 2.0 architectural model is the Application/Object Store, an internal object database for managing all aspects of an application: data objects of any medium; object properties, procedures, and methods; object classes; instances; page and window definitions; navigational information; and even applications themselves.

The Application/Object Store supports multiuser or multiteam development with tools for version control, check-in and check-out privileges, and configurable access control over all application objects. It also lets networked users concurrently access applications without copying or downloading, greatly reducing local storage requirements.

Gain*Momentum* 2.0's Application/Object Store model offers a unique capability among application development tools – an object database working in tandem with a relational database, each optimized for managing different elements of an application. By allowing developers to integrate all forms of information seamlessly, Gain*Momentum* 2.0 paves the way for a new generation of front office and extended office applications that will take advantage of client/server architecture and increase users, productivity across the entire enterprise.

Conclusion: A Built-In Future

Today's most innovative organizations recognize a unique opportunity: to rightsize legacy systems to client/server architecture while simultaneously re-engineering applications to increase productivity across new, broader audiences of end users – both internal and external. For maximum productivity, these applications must model the look and feel of the business environment, interoperate freely with external data sources, and be flexible enough to integrate all forms of information relevant to the business.

Sybase Gain*Momentum* 2.0 provides application developers with a comprehensive software architecture in which to build and deploy such applications. Utilizing a powerful object-oriented 4GL, a full suite of extensible multimedia development tools, a direct interface to RDBMS data, and a built-in object database, Gain*Momentum* 2.0 delivers a highly productive environment for developing the full range of client/server applications that support both today's enterprise and tomorrow's.

For more information and a demonstration, please call the Sybase office nearest to you.

Sybase, Inc.
Corporate Headquarters
6475 Christie Avenue
Emeryville, California
USA 94608
800-8-SYBASE
510-596-3500
Fax: 510-658-9441

European Headquarters
Sybase Europe B.V.
Planetenbaan 25
3606 AK Maarssen
The Netherlands
31-3465-82999
Fax: 31-3465-52884

Sybase Professional Services
77 South Bedford Street
Burlington, Massachusetts
USA 01803
617-238-6100
Fax: 617-270-4158

Doncastle House, Doncastle Road
Bracknell,
Berkshire RG12 4PQ
United Kingdom
44-344-360101
Fax: 44-344-360606

Europe
Belgium
Sybase Software SA/NV
32-2725-5171
Fax: 32-2725-6550

France
Sybase France SARL
33-1-41-07-8888
Fax: 33-1-41-07-8800

Germany
Sybase GmbH
49-211-59760
Fax: 49-211-5976-111

Italy
Sybase Italia S.R.L.
39-55-340560
Fax: 39-55-340558

The Netherlands
Sybase B.V.
31-3465-82999
Fax: 31-3465-52884

Spain
Sybase Iberia, S.A.
34-1-302-0900
Fax: 34-1-320-8438

Switzerland
Sybase AG
41-42-32-1277
Fax: 41-42-31-3120

United Kingdom
Sybase (UK) Limited
44-628-597100
Fax: 44-628-28844

Distributors in:
Cyprus
Denmark
Finland
Greece
Hungary
Ireland
Israel
Norway
Portugal
Sweden
Turkey

Canada
Sybase Canada Limited
416-566-1803
Fax: 416-566-1806

Northern Asia
Japan
Sybase K.K.
Sumitomo Fudosan
Sarugaku-cho Building 14F
2-8-8 Sarugaku-cho
Chiyoda-ku, Tokyo 101
Japan
81-3-5280-1141
Fax: 81-3-5280-1161

Distributors in:
Hong Kong
Korea
Taiwan

AustralAsia
Distributors in:
Australia
New Zealand
Singapore (S.E. Asia)

Latin America
Distributors in:
Argentina
Brazil
Chile
Costa Rica (Central America)
Ecuador
Mexico
Panama
Venezuela

For other Asia Pacific and Latin America inquiries, please contact Sybase Intercontinental Operations in the USA:
510-596-3500
Fax: 510-596-3446

8848.0693

Handouts of

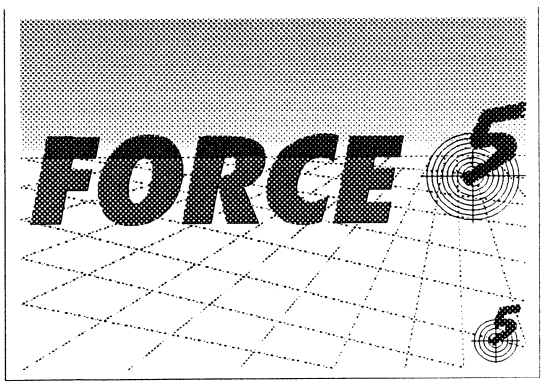
Presentation of

FORCE-5

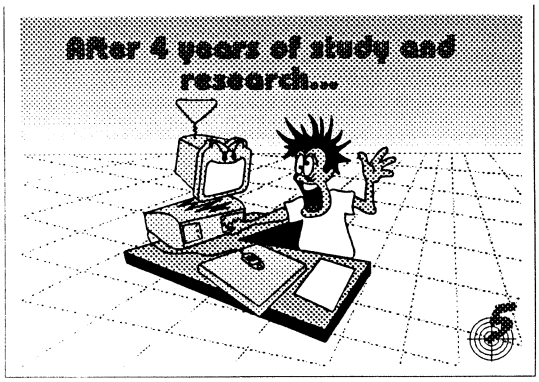
The Object Oriented
Application Development System

By R.A. Postma

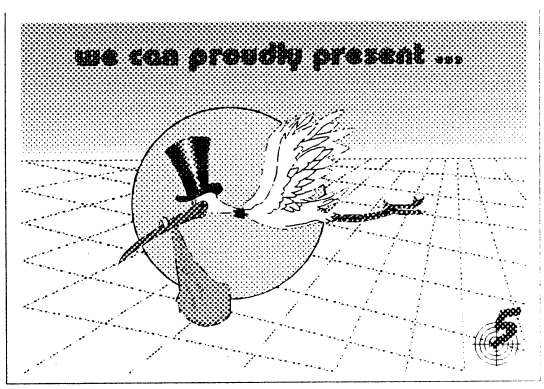
FORCE 5



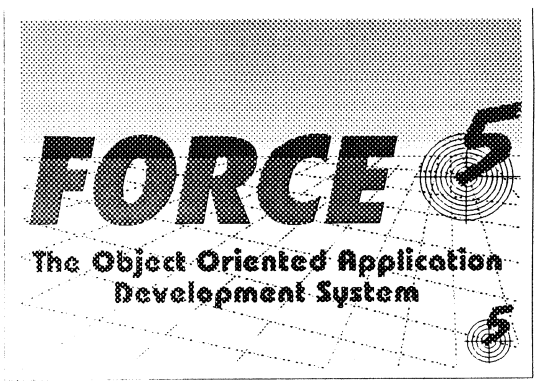
Slide 1



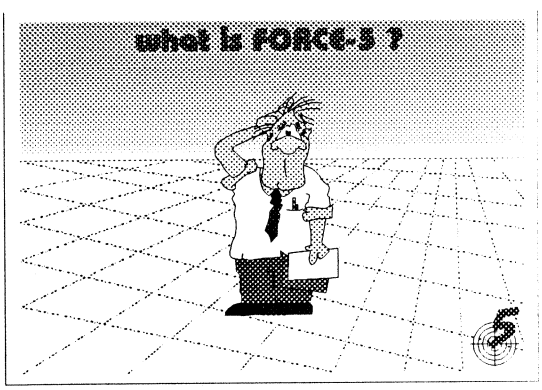
Slide 2



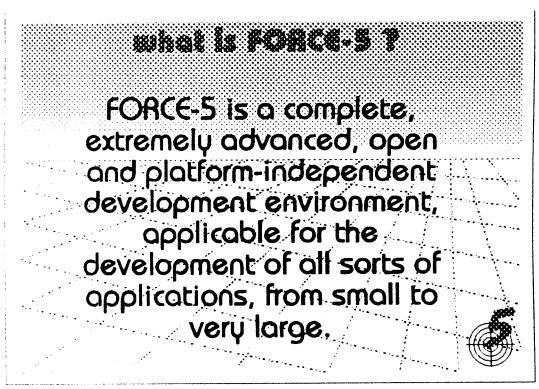
Slide 3



Slide 4

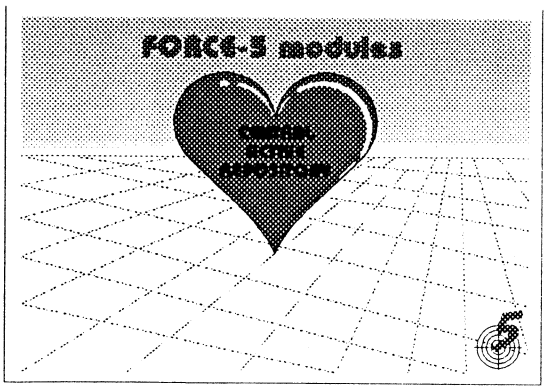


Slide 5

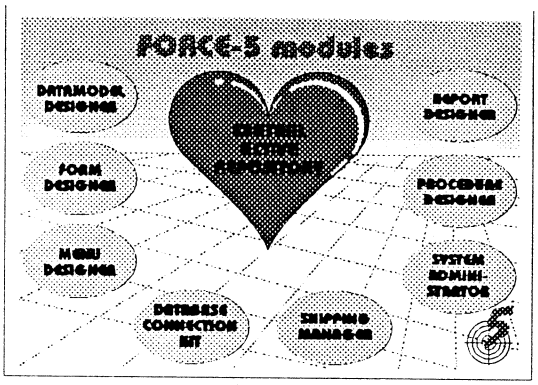


Slide 6

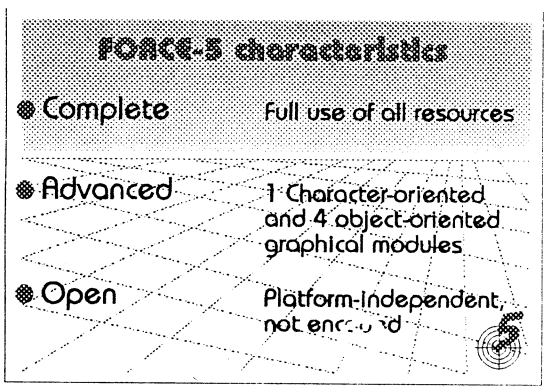
FORCE 5



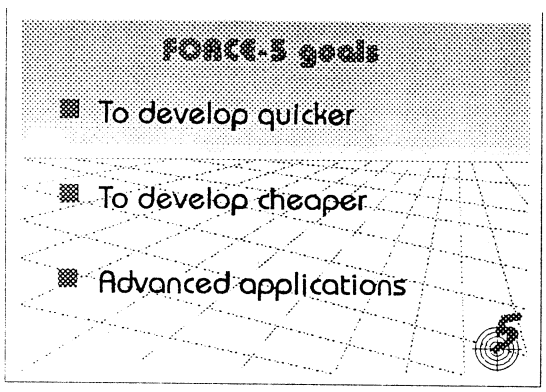
Slide 7



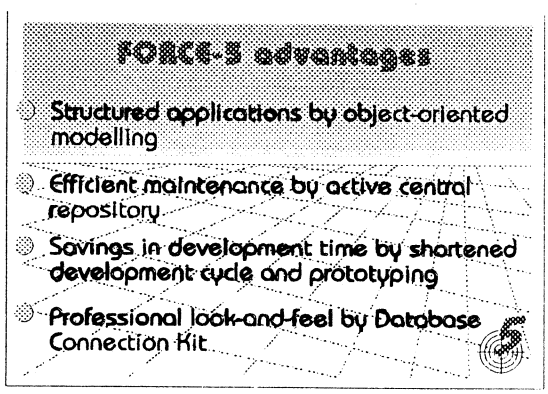
Slide 8



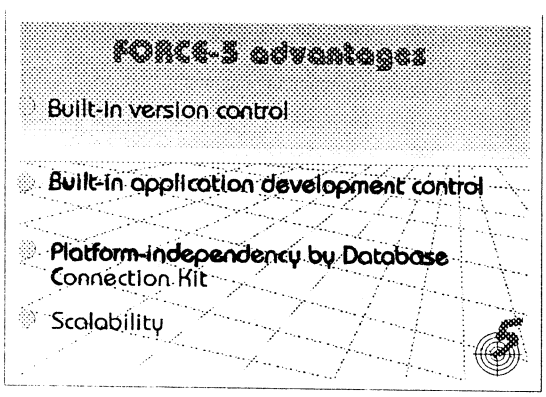
Slide 9



Slide 10



Slide 11



Slide 12

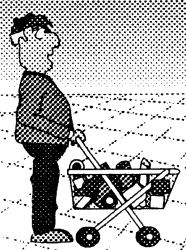
FORCE 5

FORCE-5 advantages

- Up-to-date: object-oriented, client-server, RDBMS
- Multi-lingual
- Heterogene applications
- Extensive application documentation facilities

Slide 13

FORCE-5: why in the market?



Slide 14

FORCE-5: why in the market?

After a thorough study in 1988, it appeared, that none of the development environments present at that time, met the following demands:

Slide 15

FORCE-5: why in the market?

1. Complete development of an application without the influence of technical dependencies.
2. Full support of the development activities by automized tools.
3. All application information stored at a central place and up-to-date.
4. Control of the functionality of the application by the user in an early stage by prototyping.

Slide 16

FORCE-5: why in the market?

The FORCE-5 development environment does meet all these demands:

where other development environments stop, FORCE-5 goes on

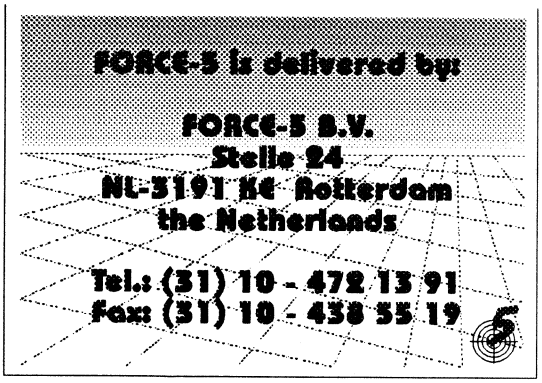
Slide 17

FORCE-5: in short

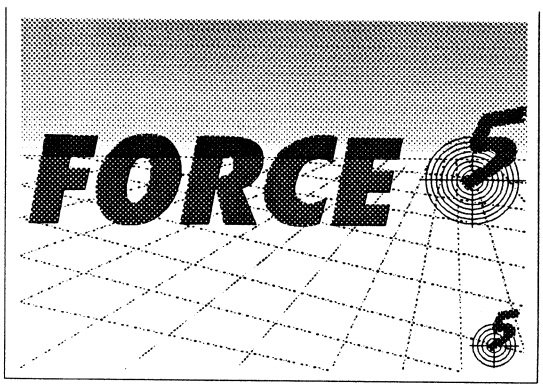
all you need is ...

FORCE-5

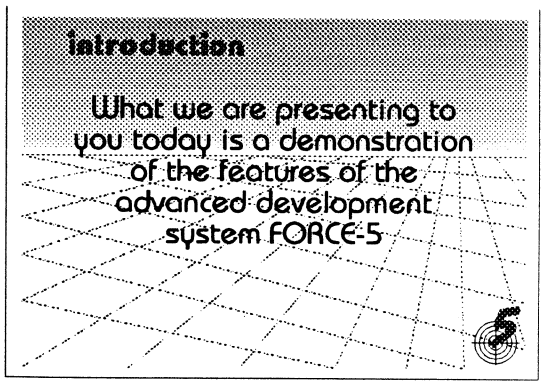
Slide 18



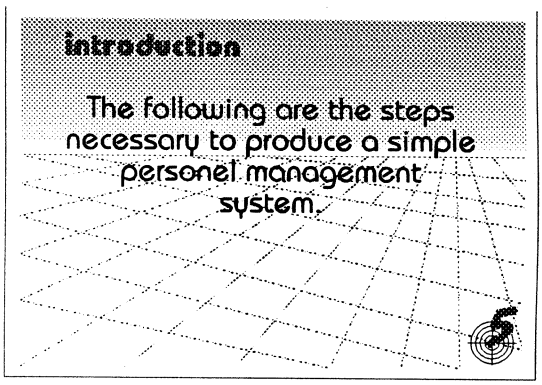
Slide 19



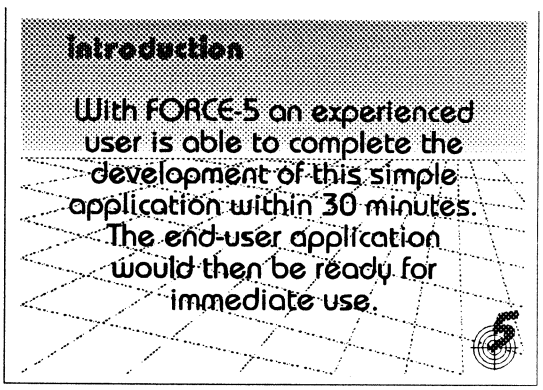
Slide 20



Slide 21

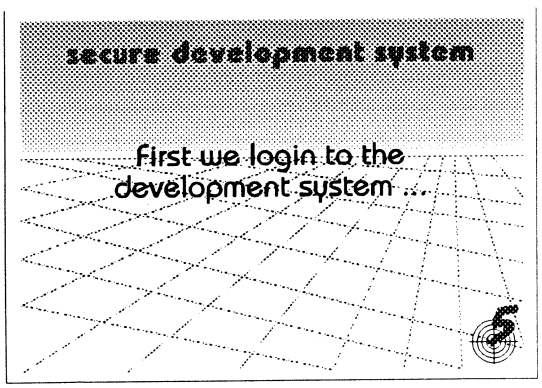


Slide 22

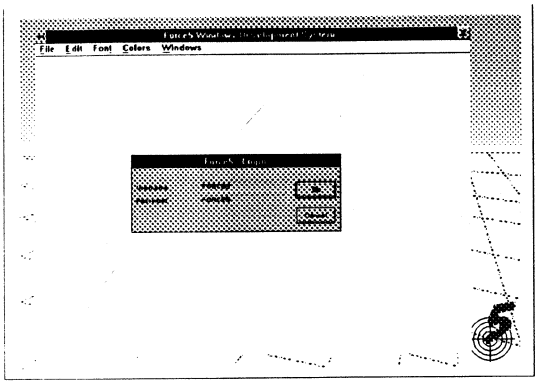


Slide 23

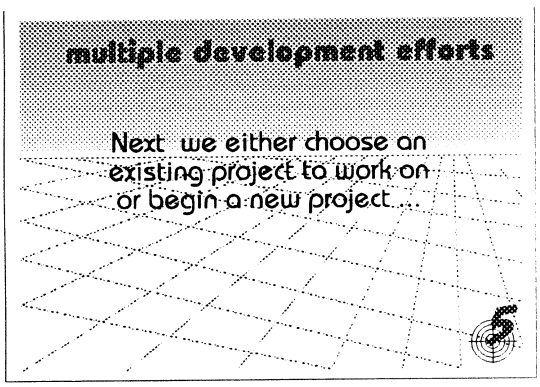
FORCE 5



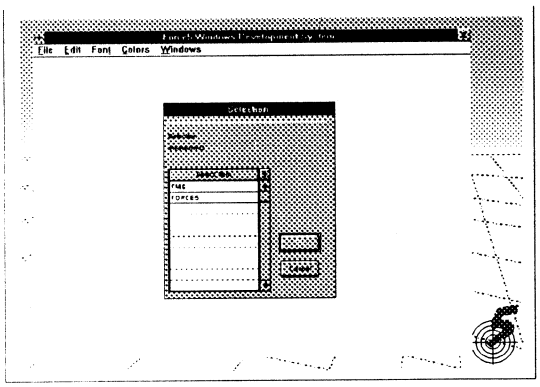
Slide 24



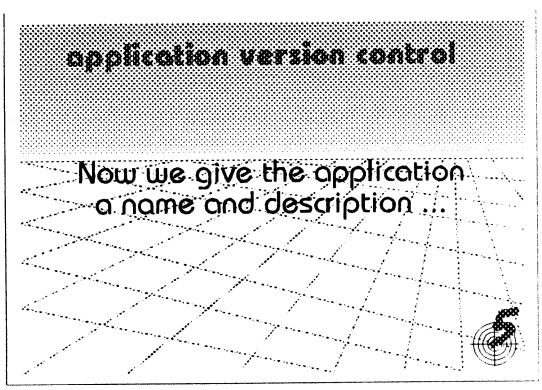
Slide 25



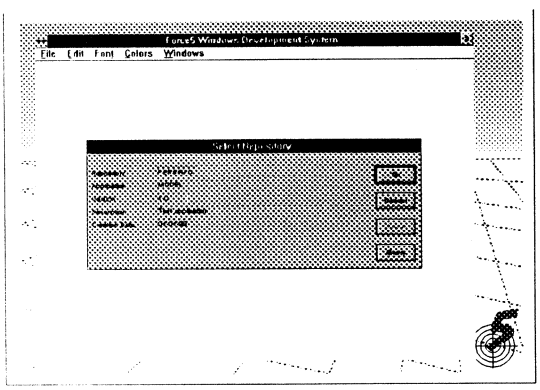
Slide 26



Slide 27



Slide 28



Slide 29

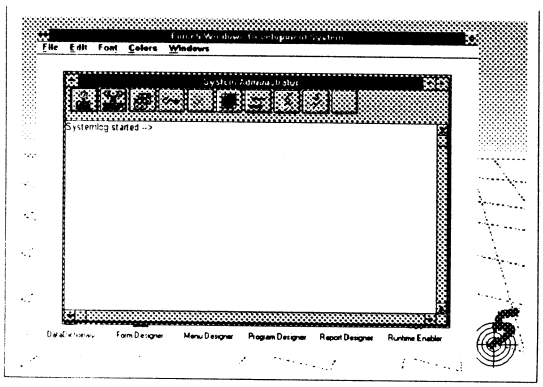
FORCE 5

built-in system administrator

The first window to start up is the system administrator window.

Here all work is logged and access is controlled and monitored.

Slide 30

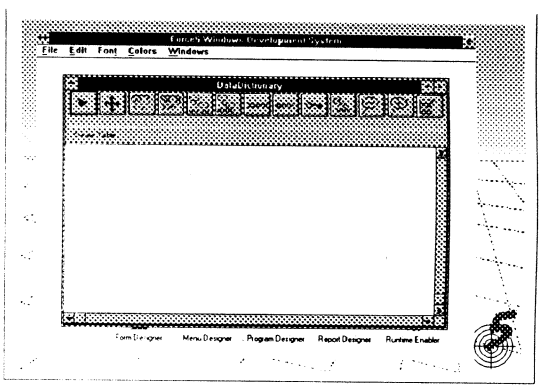


Slide 31

data dictionary designer

To create a new data model we go to the data dictionary designer and click on: table new...

Slide 32



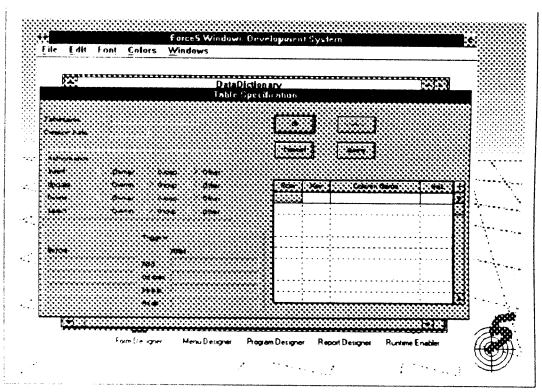
Slide 33

specify table rows

After clicking 'table new' we get the 'table specification screen'.

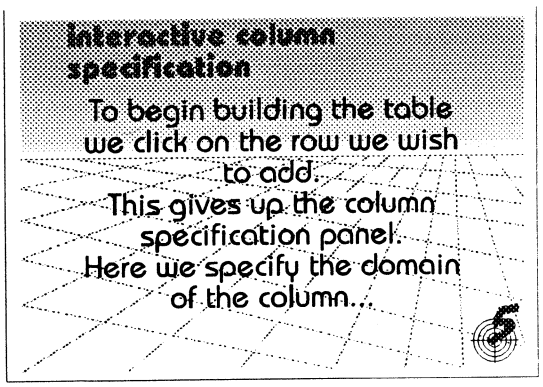
Here we will specify the columns, rows and domains that make up an entry in the data dictionary. First name the table and then add authorizations and triggers...

Slide 34

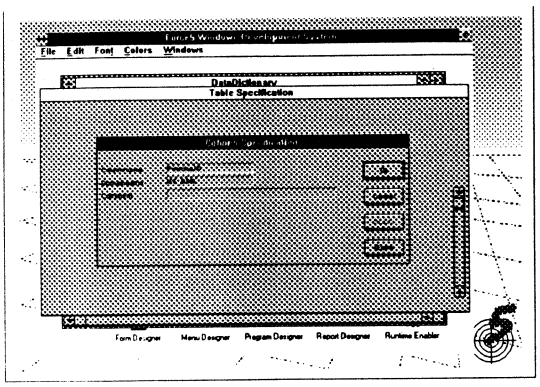


Slide 35

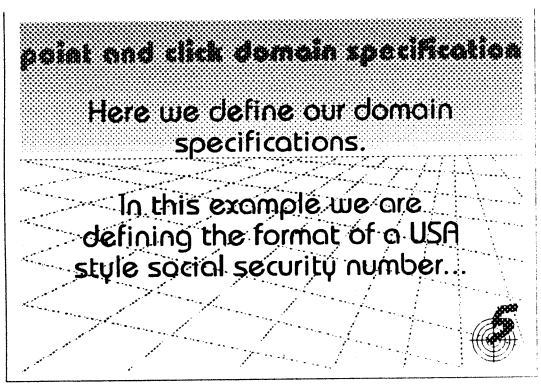
FORCE 5



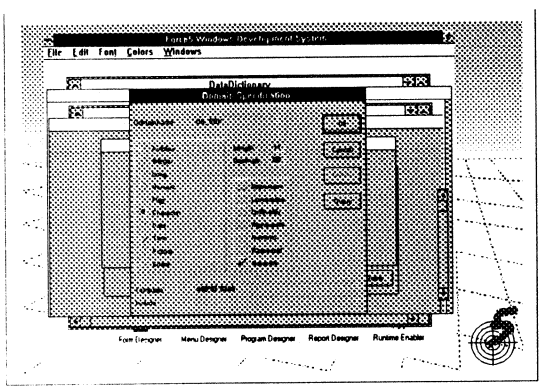
Slide 36



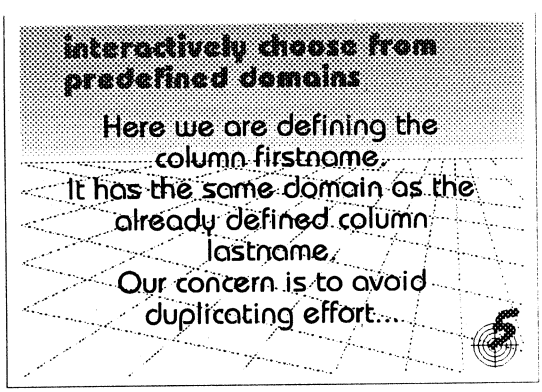
Slide 37



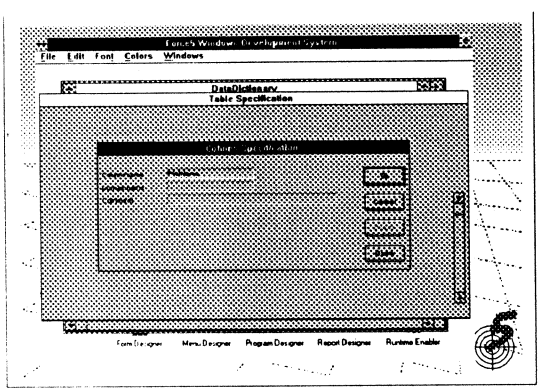
Slide 38



Slide 39



Slide 40



Slide 41

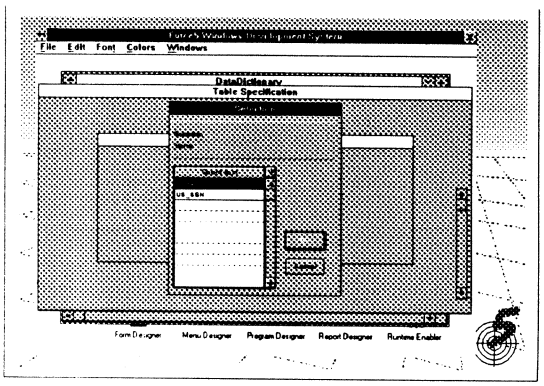
FORCE 5

interactively choose from predefined domains

We can click on the existing definition of the domain 'name' for our column.

Then we just click on OK...

Slide 42



Slide 43

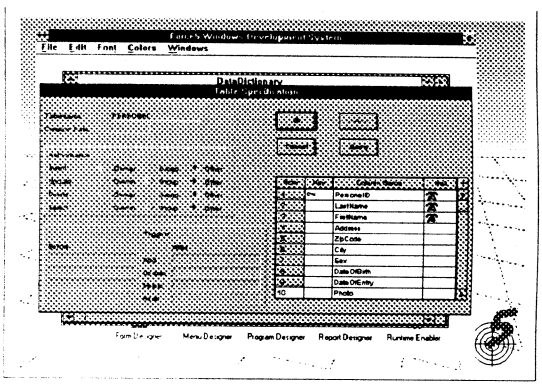
a finished table

Continuing the previous process for the remaining rows we soon have the rest of the table defined.

Rows can be marked with a 'key' to indicate that they are keys in the database.

Rows can also be marked with 'bells' to indicate that they are manditory entries for a valid table...

Slide 44



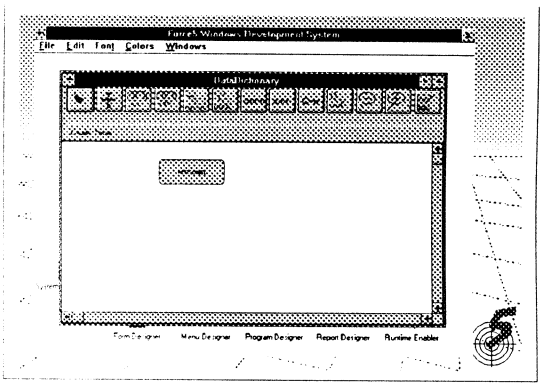
Slide 45

graphical layout of data dictionary

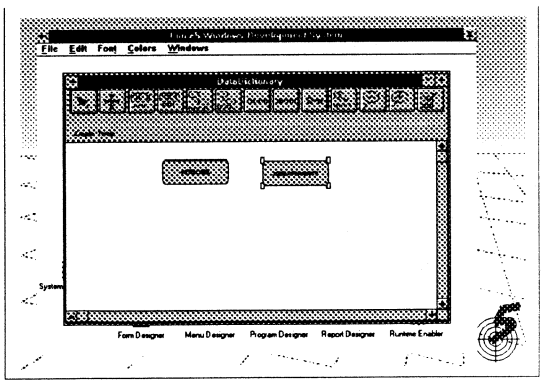
As we finish defining each of our tables we then lay them out on to the data dictionary workspace.

This is in preparation of defining our data model relationships...

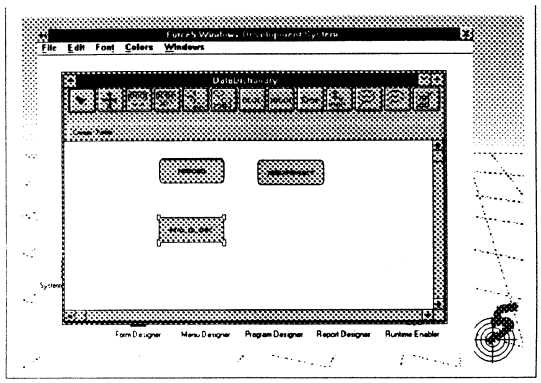
Slide 46



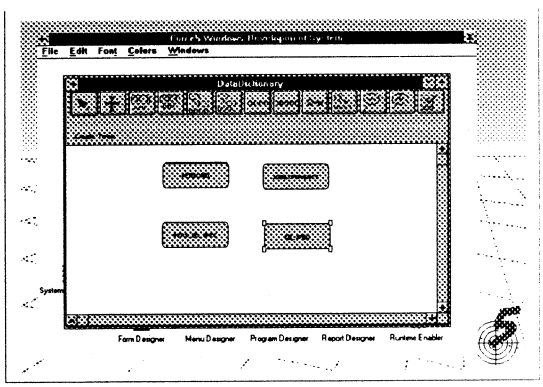
Slide 47



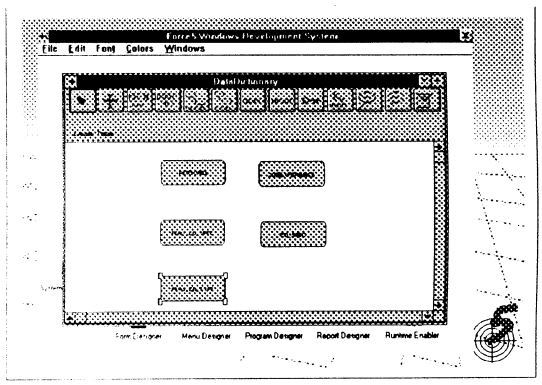
Slide 48



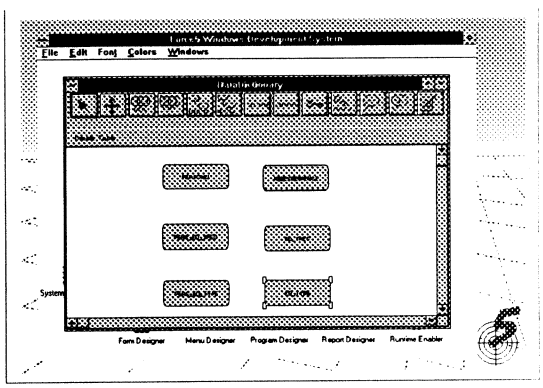
Slide 49



Slide 50

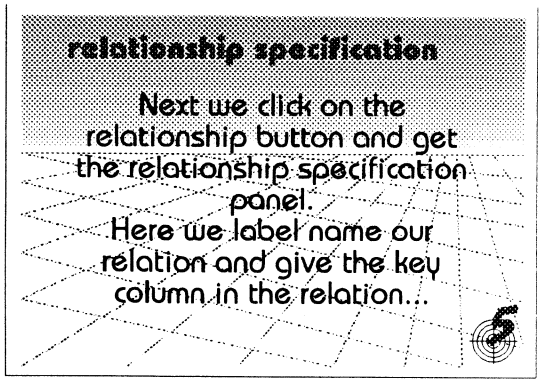


Slide 51

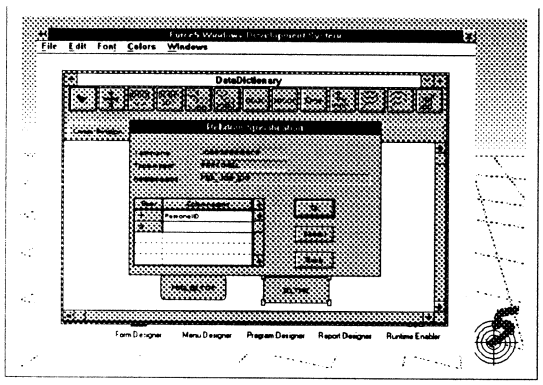


Slide 52

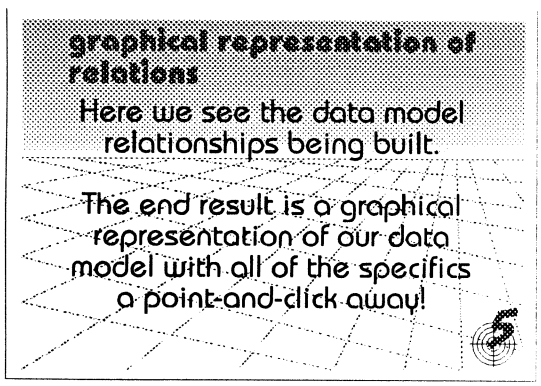
FORCE 5



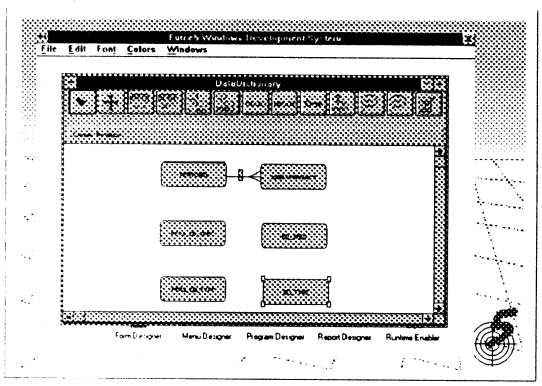
Slide 53



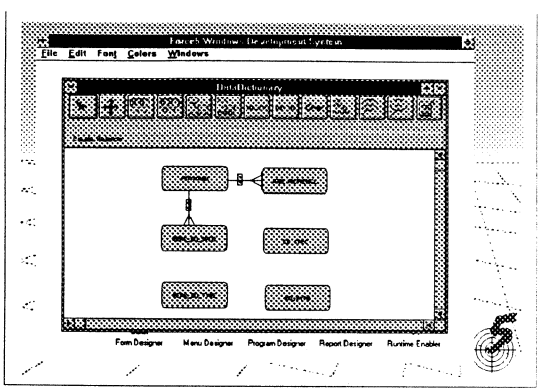
Slide 53



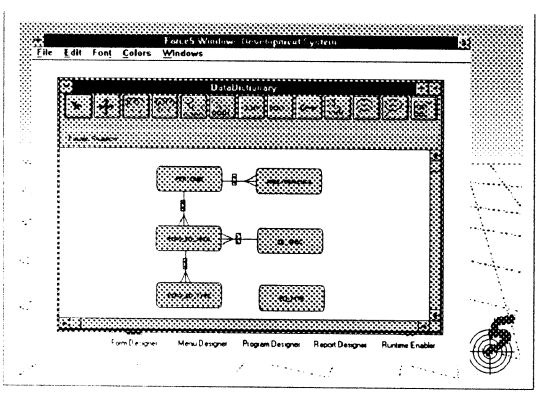
Slide 55



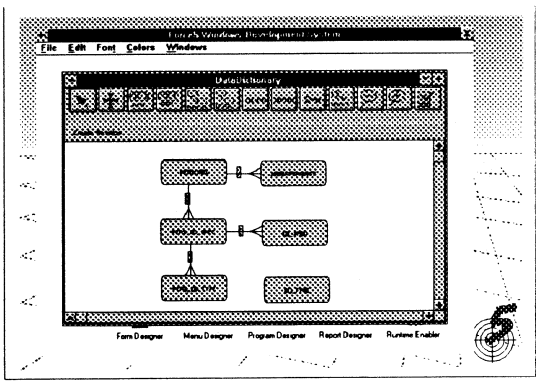
Slide 56



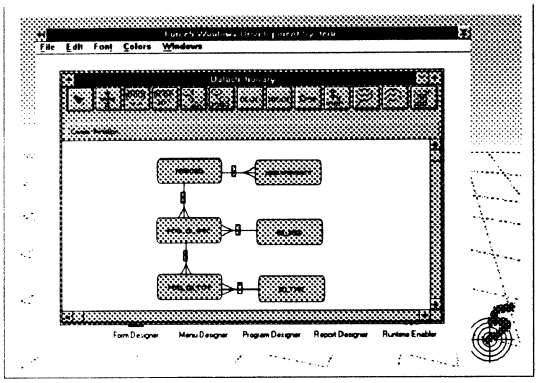
Slide 57



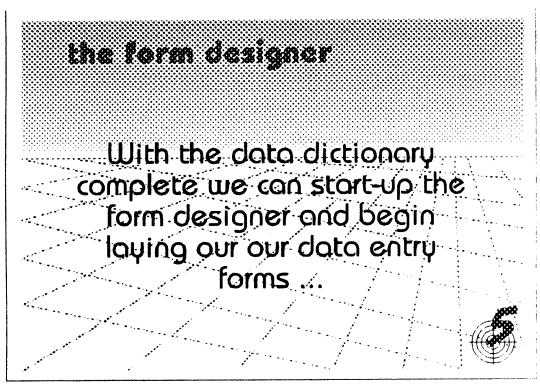
Slide 58



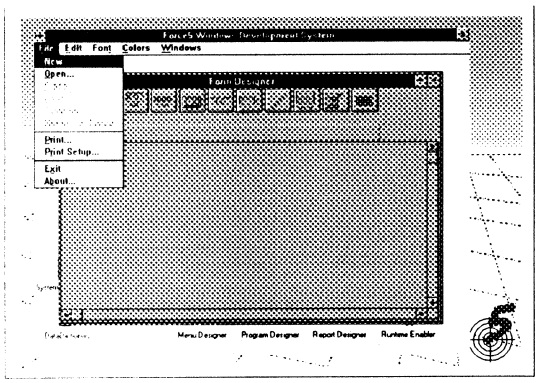
Slide 59



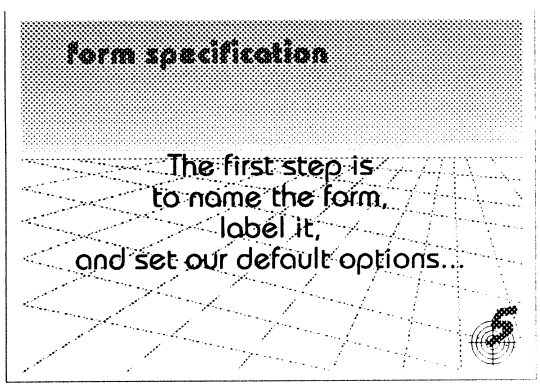
Slide 60



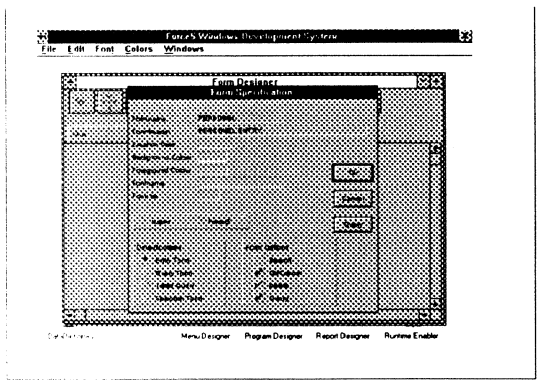
Slide 61



Slide 62



Slide 63



Slide 64

FORCE 5

default buttons placed immediately

After specifying our form
we return to our workspace
and our default buttons
are set into place.

These buttons
can later be moved
if we wish...

Slide 65

default buttons placed immediately

After specifying our form
we return to our workspace
and our default buttons
are set into place.

These buttons
can later be moved
if we wish...

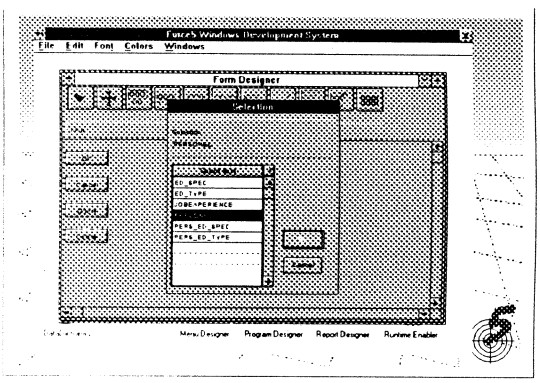
Slide 66

selecting a table for the form

Clicking on the table add
button gives us a list of
defined tables in the data
dictionary to choose from.

We click on the selection, it
appears in the selection
panel and then we click on
the OK button...

Slide 67

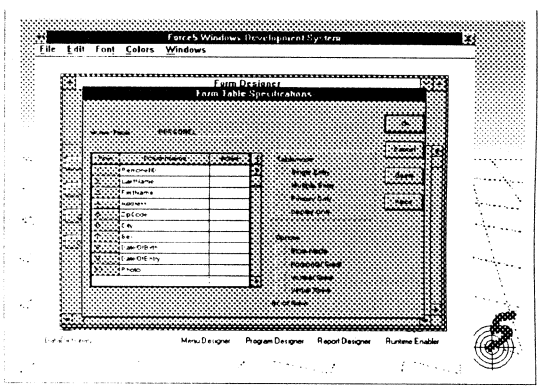


Slide 68

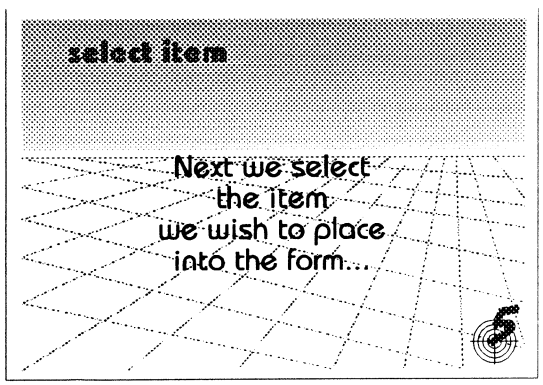
specify form entry mode

On this table
we specify
the data entry mode
of the table...

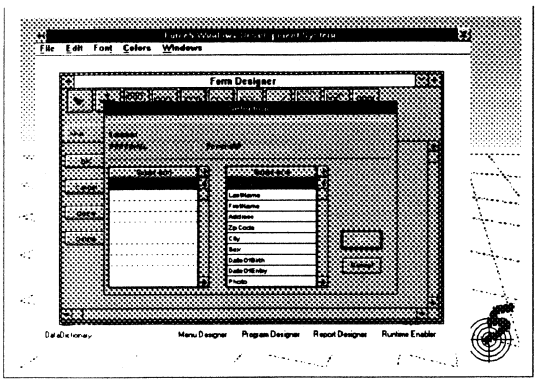
Slide 69



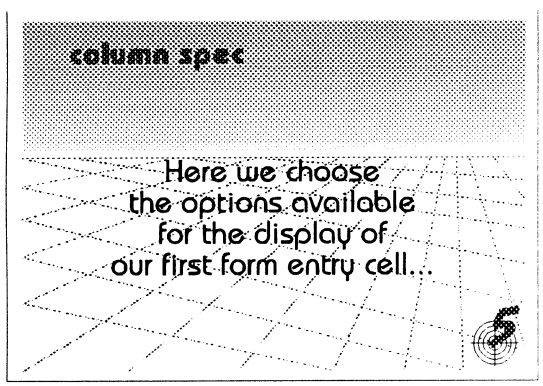
Slide 70



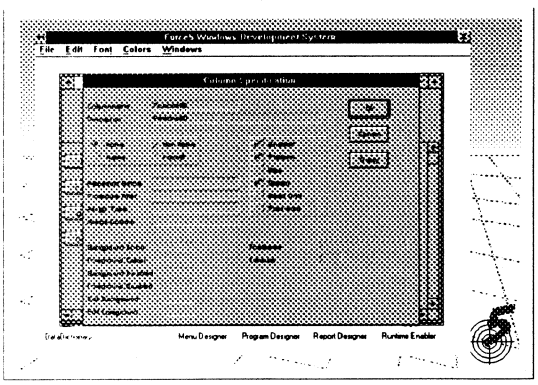
Slide 71



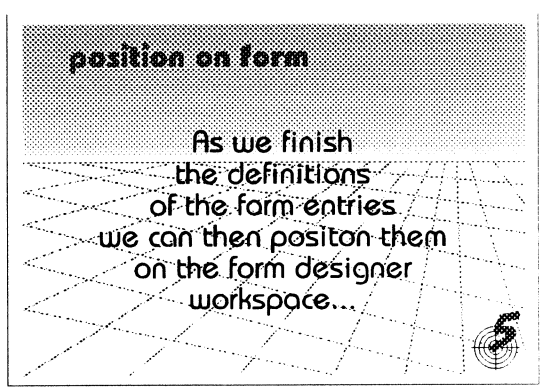
Slide 72



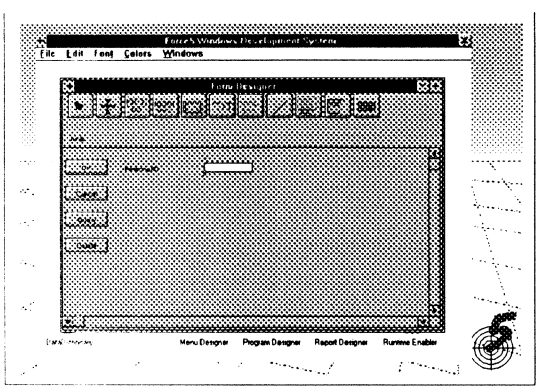
Slide 73



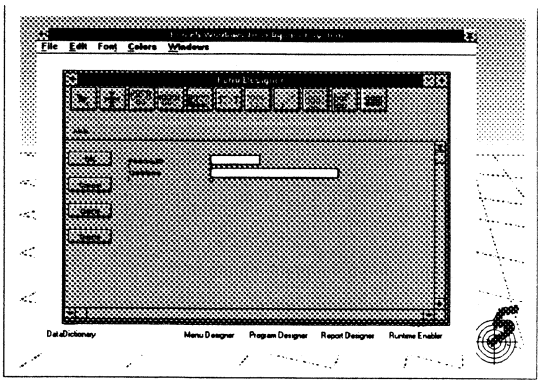
Slide 74



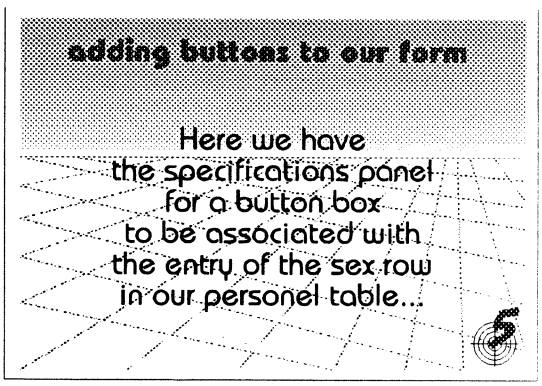
Slide 75



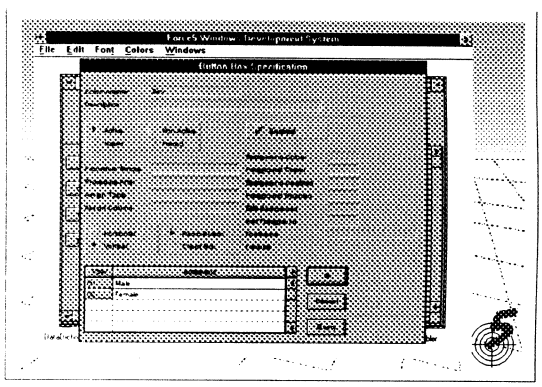
Slide 76



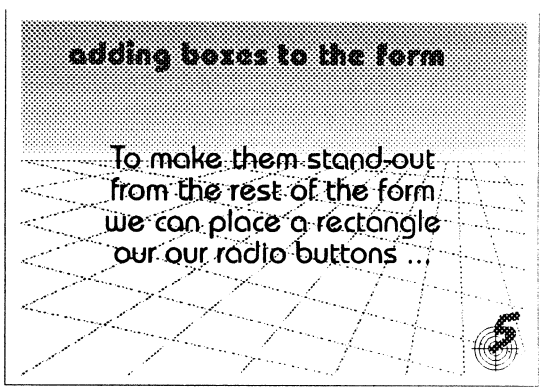
Slide 77



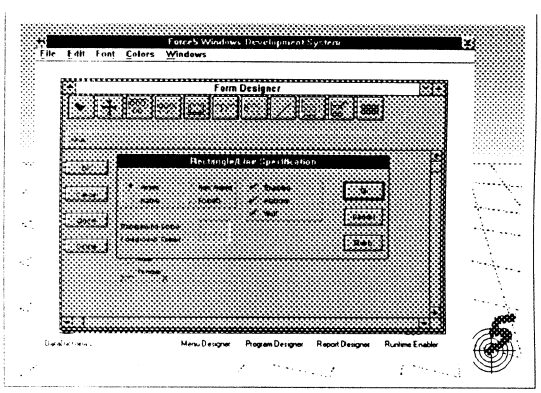
Slide 78



Slide 79

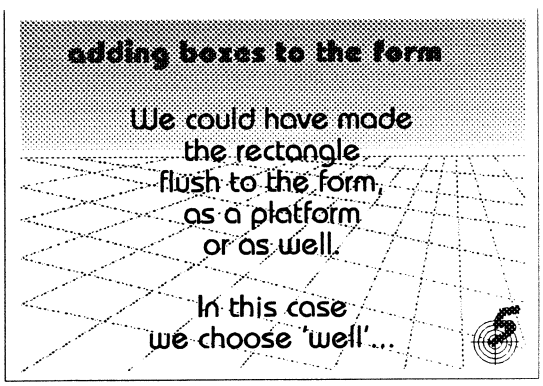


Slide 80

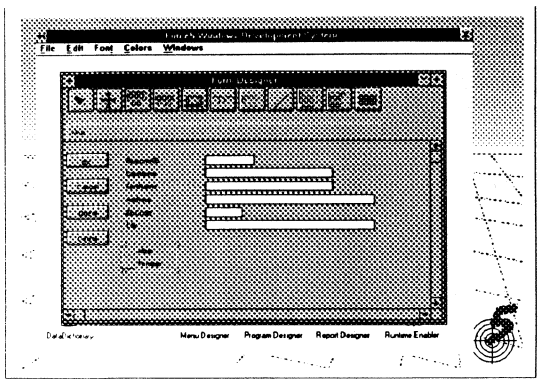


Slide 81

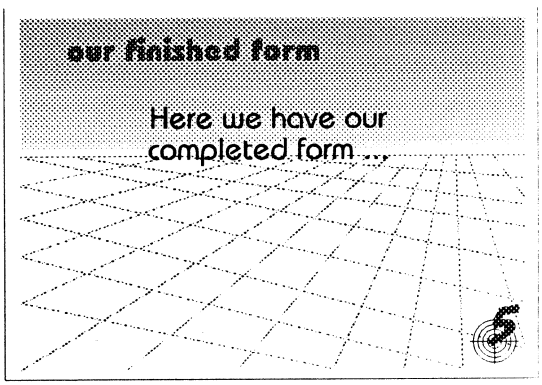
FORCE 5



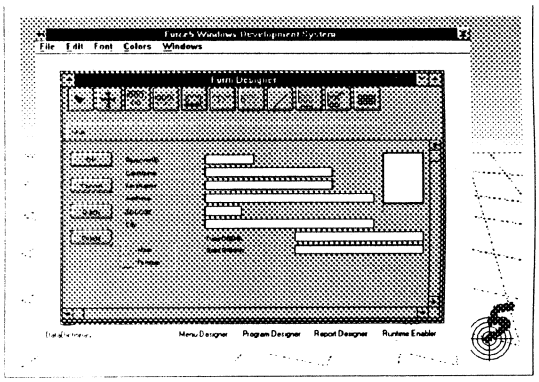
Slide 82



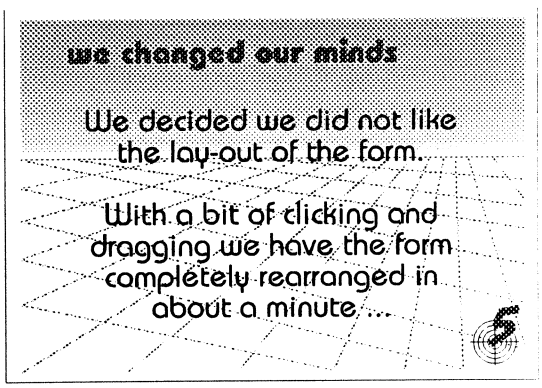
Slide 83



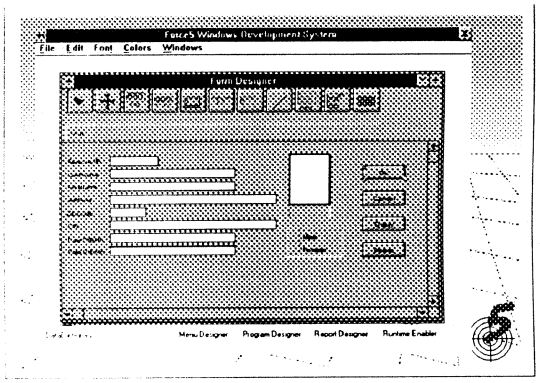
Slide 84



Slide 85

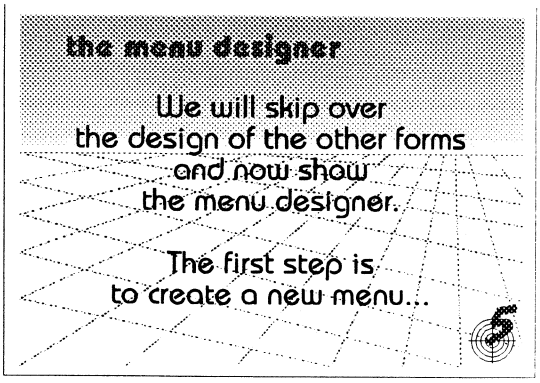


Slide 86

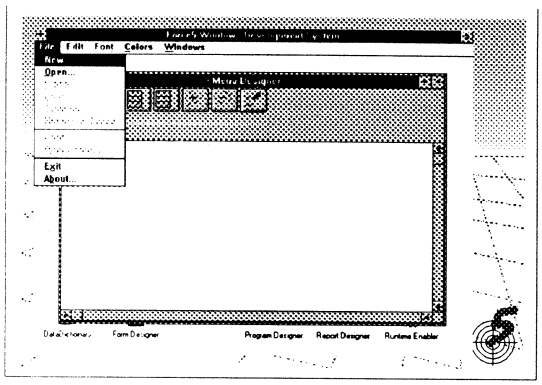


Slide 87

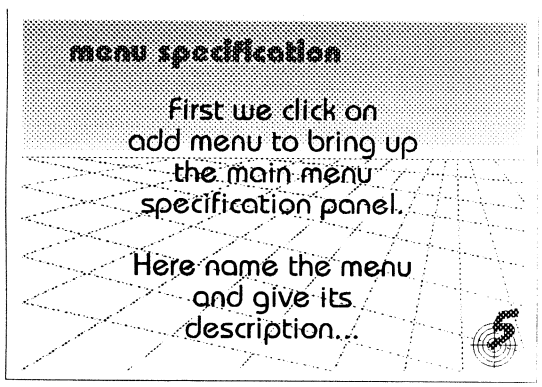
FORCE 5



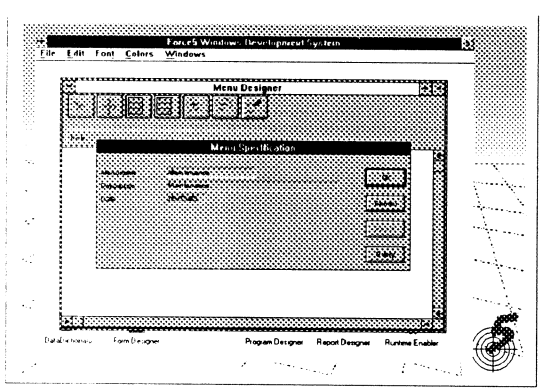
Slide 88



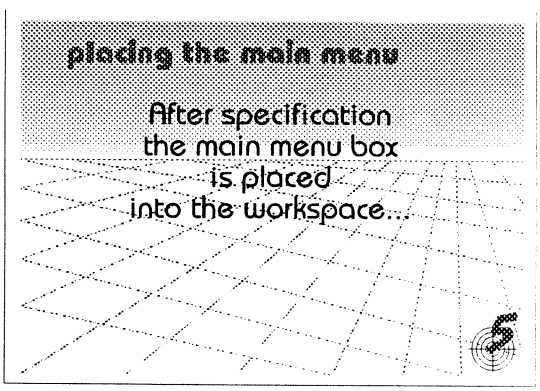
Slide 89



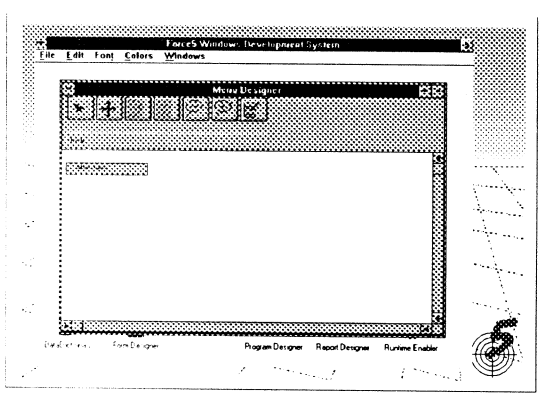
Slide 90



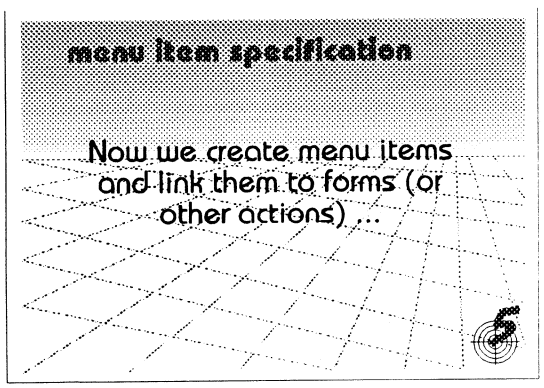
Slide 90



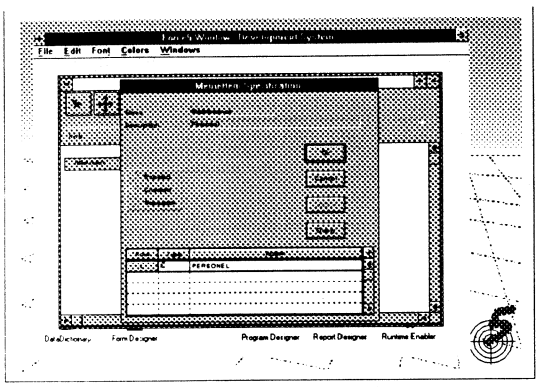
Slide 92



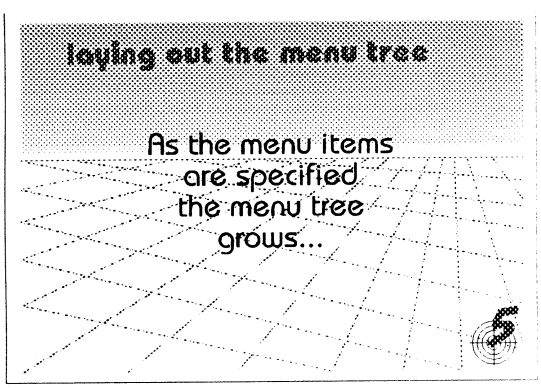
Slide 93



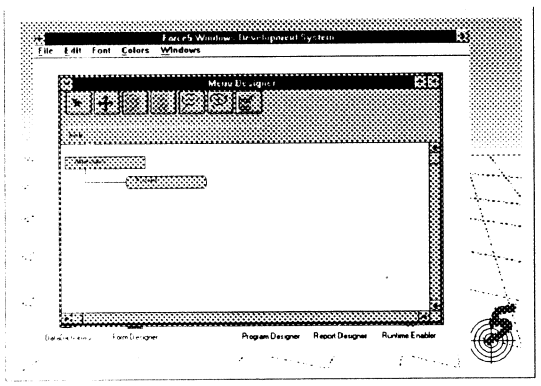
Slide 94



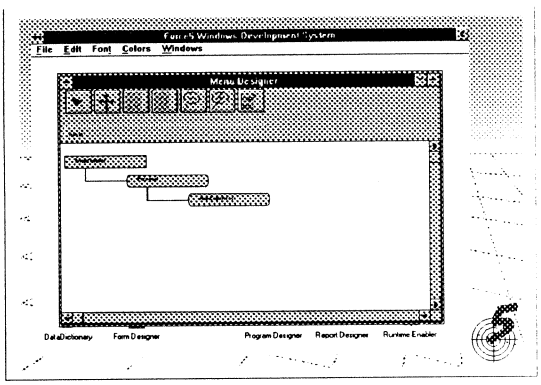
Slide 95



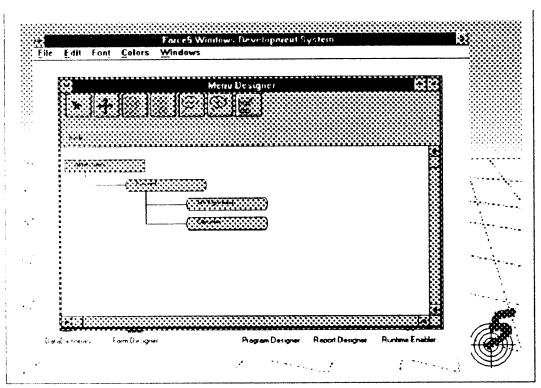
Slide 96



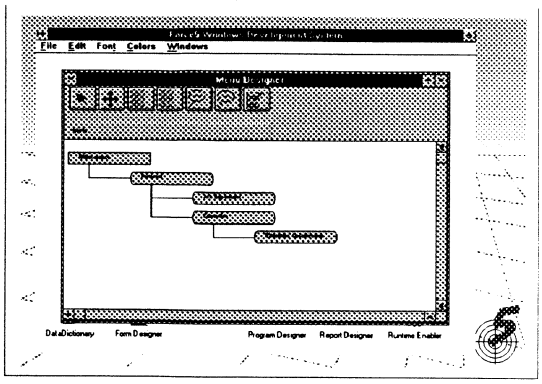
Slide 97



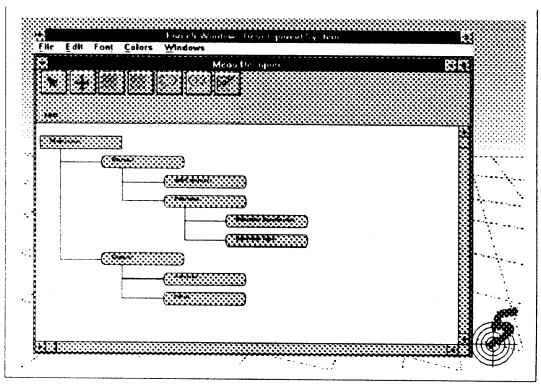
Slide 98



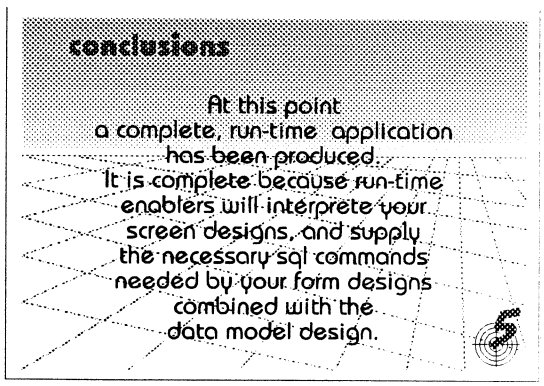
Slide 99



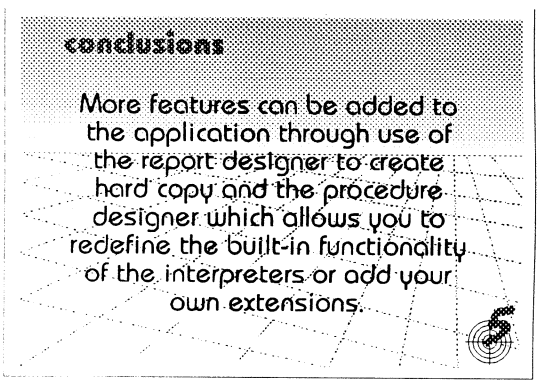
Slide 100



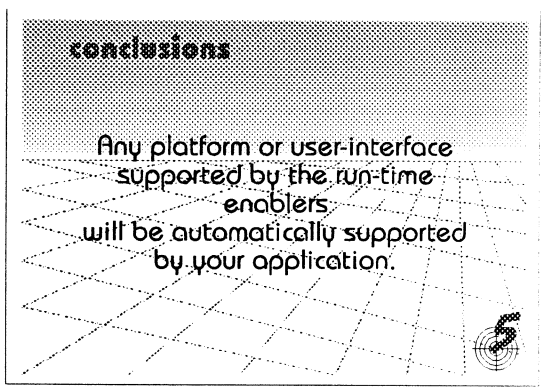
Slide 101



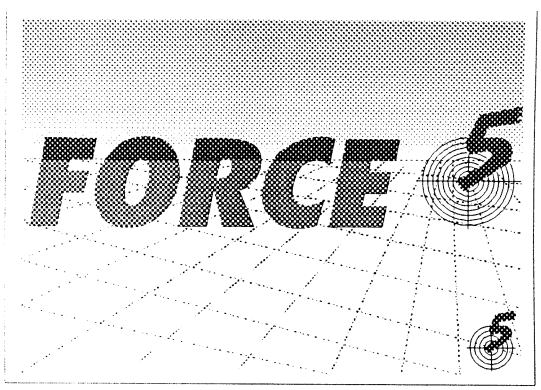
Slide 102



Slide 103



Slide 104

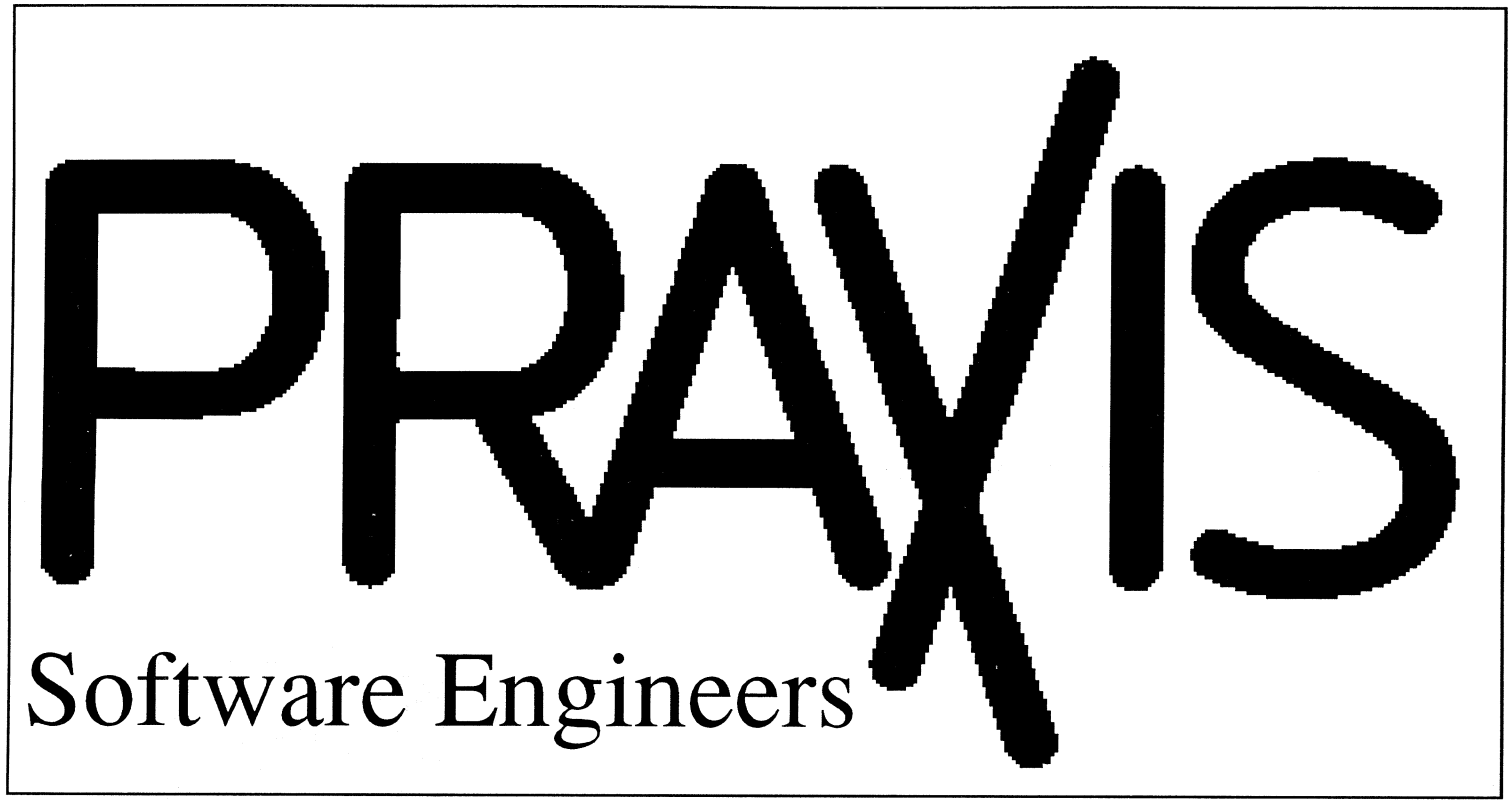


Slide 105

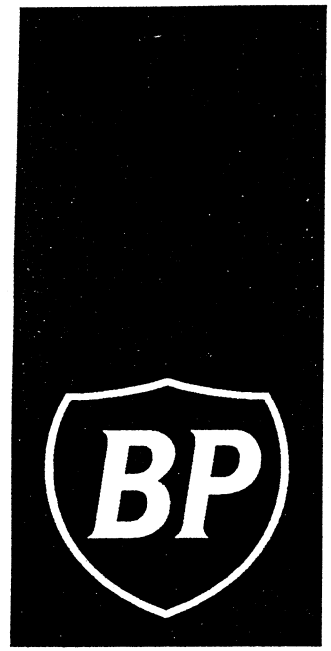
We thank you
for joining
this presentation.

For more information,
call: (31) 10 - 472 13 91.

FORCE-5 B.V.



THE SOFTWARE ENGINEERING COMPANY OF
TOUCHE ROSS & CO



FORPPAS Project

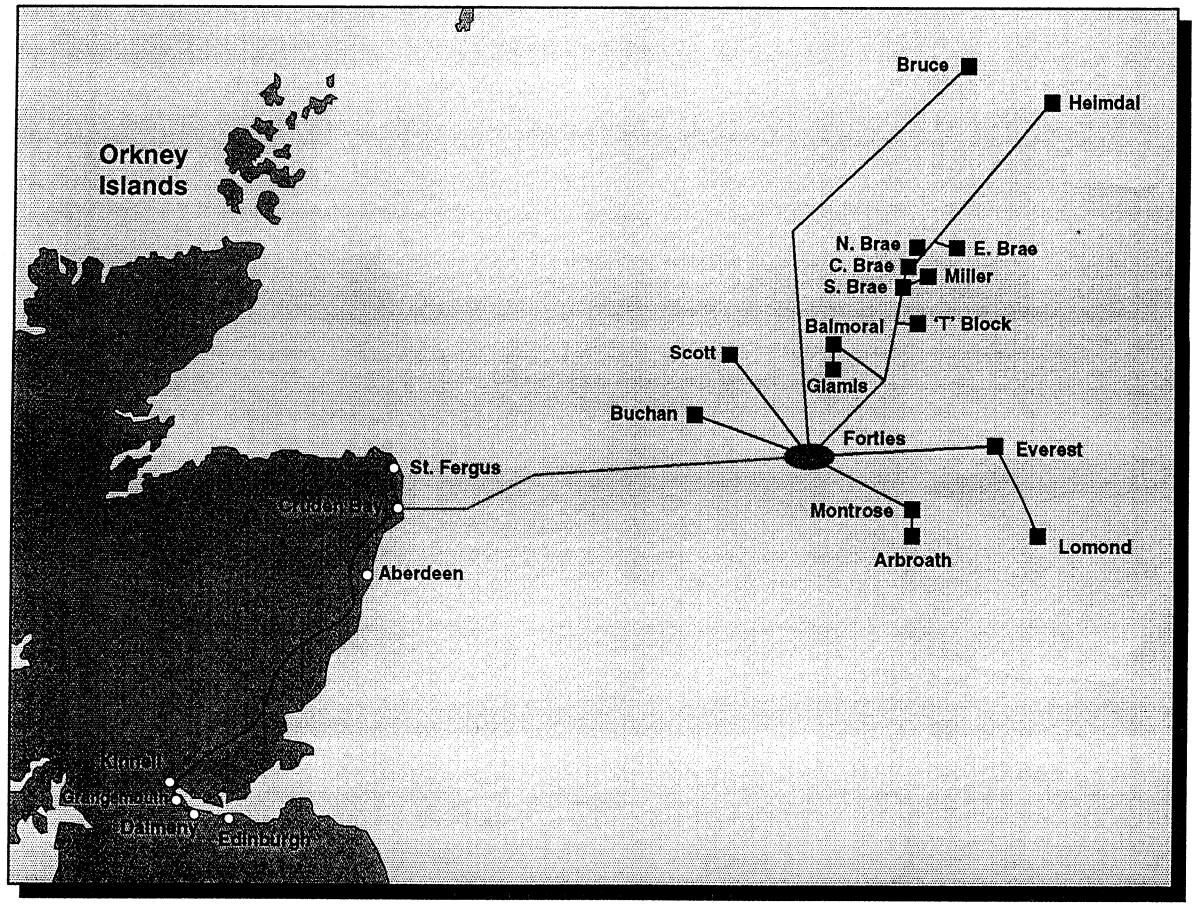
Forties Pipeline Production & Allocation System

BP Exploration

S. Ellis, Aug 1993, 01105549

Forties Pipeline System

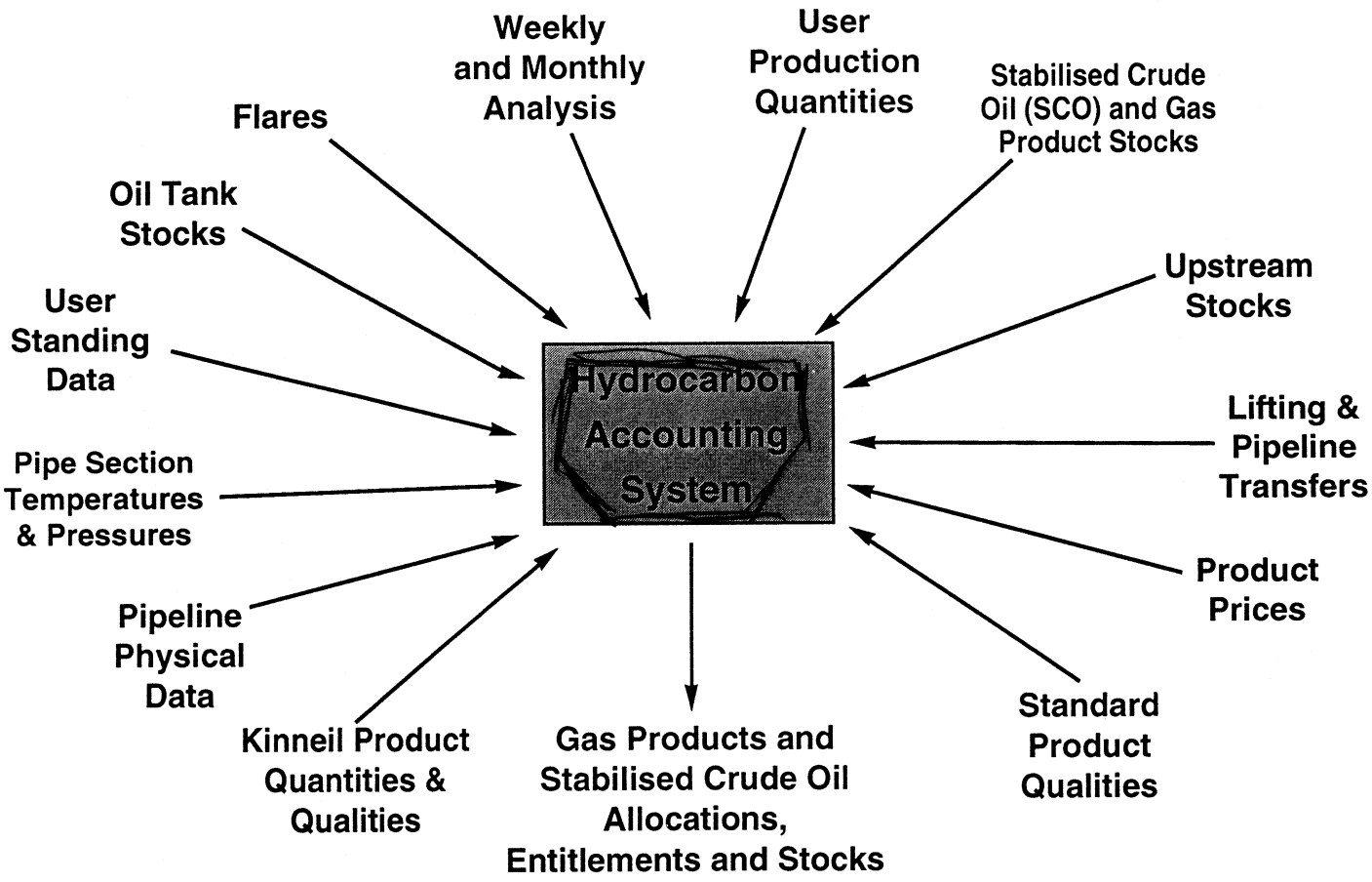
Fields Using the Forties Pipeline System



S. Ellis, Aug 1993, 01105552a

Forties Pipeline System

Data Requirements for Hydrocarbon Accounting



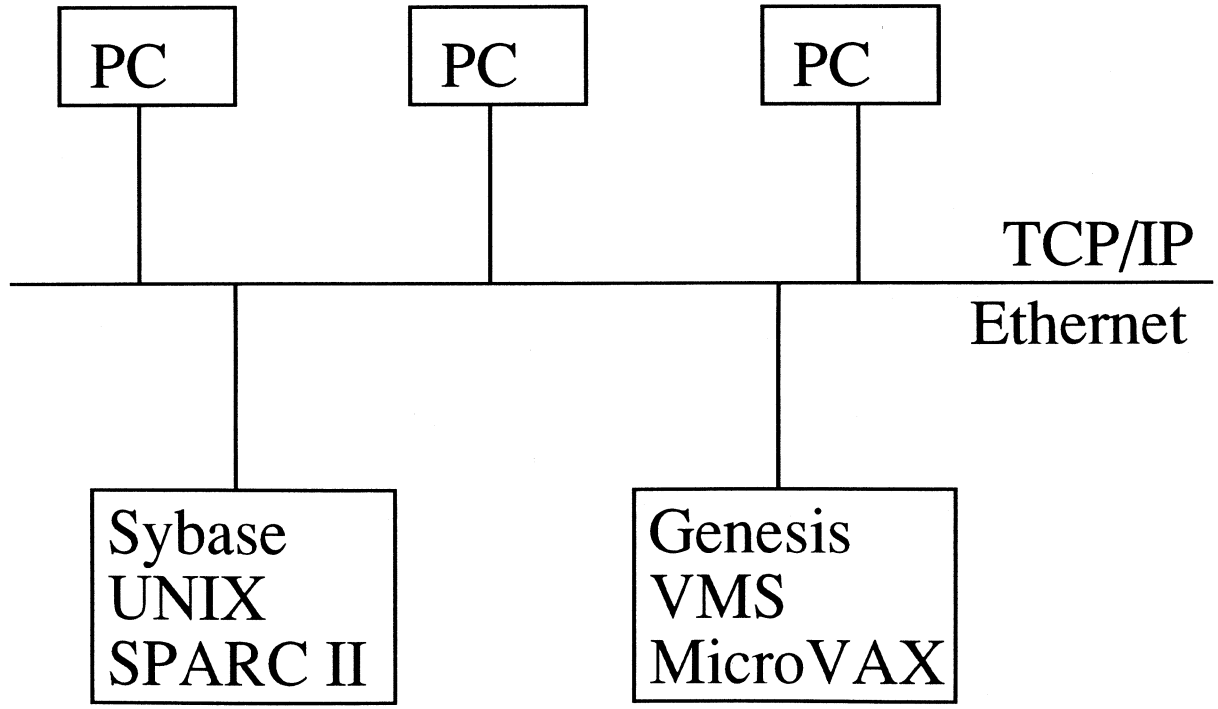
S. Ellis, Aug 1993, 01105551

Agenda

- Project Hardware and Software
- Assumptions And Traditional Design
- Praxis' 1st, 2nd and 3rd Designs
- Data Models And Detailed Spreadsheet Design
- Sybase, Three Steps To Easy Interaction
- Conclusion

PRAXIS

Project Hardware

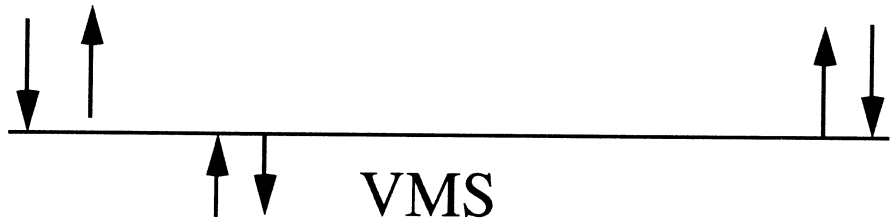


PRAXIS

Software and Connectivity

PC
Excel for Windows
?
Lan Manager

UNIX
Sybase
?
LM Server on UNIX



VMS
TGV MULTINET on VMS

Praxis Control Software
Genesis

PRAXIS

The Missing Link

- Why Not Q&E ?
- SequeLink Chosen

PRAXIS

SequeLink

- Seamless Interface To Excel
- Execute Transact-SQL
- SequeLink Has Own Data Buffers

PRAXIS

SequeLink To Excel

- En Mass Via Clipboard
- Single Column Via Clipboard
- Single Cell Direct From SequeLink Buffer

PRAXIS

Spreadsheet Design Assumptions

- Spreadsheet Design =

  Logical Data Model

  + Calculated Attributes

  + Formula

- Spreadsheets Would Be Self-Documenting

PRAXIS

Traditional Spreadsheet Design

- Same Area Is Both Calculator And Presenter
- Spreadsheet Power Allows Concentration On Aesthetics
- User Can Develop Own Applications

PRAXIS

Implications Of Traditional Design

- Easily Understood By User
- Difficult To Interface To Database
- Maintenance Problems
- Expects Fixed Data Volumes

PRAXIS

Traditional Approach

Presentation	Excel
Calculation	Excel
Data Placement	N/A
Data Extraction	N/A
Data Storage	N/A

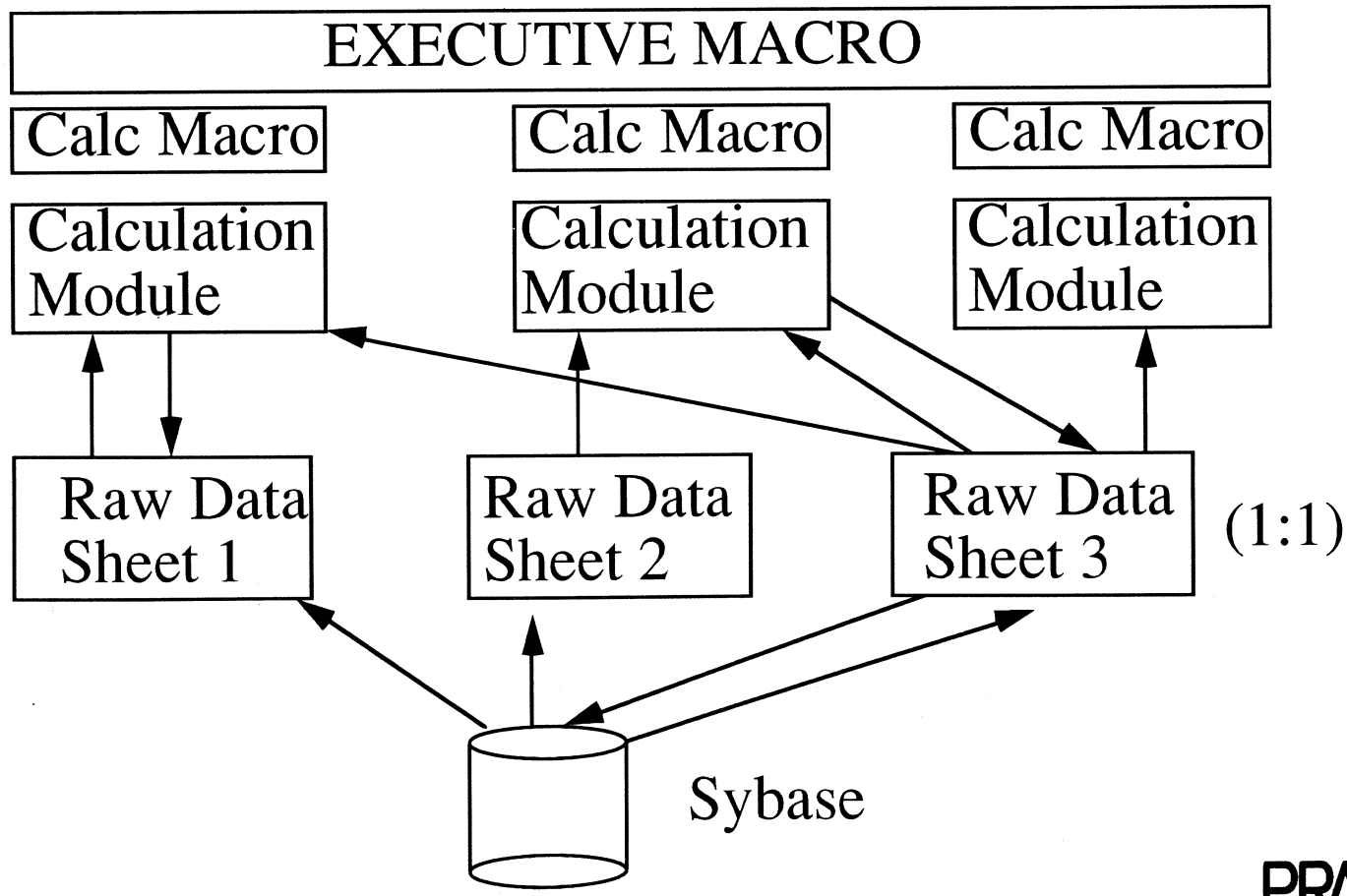
PRAXIS

Problems

- Layout Not Suitable For Database Interface
- Cannot Cope With Changing Data Volumes

PRAXIS

First Design Refinement



PRAXIS

Problems Addressed

- ❑ ✓ Layout suitable For Database Interface (1:1)
- ❑ ✓ Can Cope With Changing Data Volumes

Additional Advantages

- ❑ ✓ Independent Module Development
- ❑ ✓ Can Handle Large Data Volumes
- ❑ ✓ Locating Specific Calculations Is Easy

PRAXIS

Raw Data Sheets

Presentation	Excel
Calculation	Excel
Data Placement	Excel
Data Extraction	Excel Sequelink and Sybase
Data Storage	Sybase

PRAXIS

Problems Remaining Or New

- (*) Database Structure <> Calculation Requirements
- (*) Excel Is Not Good At Data Joins
- (*) Large Code Overhead - Slow, Error Prone
- Data Duplication
- Explicit Sequencing Of Calculations
- (*) Inappropriate Use Of Products

PRAXIS

2nd Design - Eliminate Excel Joins & Improve Product Use

- Raw Data Sheets Now Hold Pre-Joined Data
- Reduces Excel Code
- Better Use of Sybase

PRAXIS

Second Approach

Presentation	Excel
Calculation	Excel
Data Placement	Excel/SequeLink
Data Extraction	Sybase
Data Storage	Sybase

PRAXIS

Problems Addressed

- ✓ Database View = Calculation Requirements
- ✓ Excel Not Required To Do Data Joins
- ✓ Reduced Amount Of Excel Code
- ✓ More Appropriate Use Of Products

PRAXIS

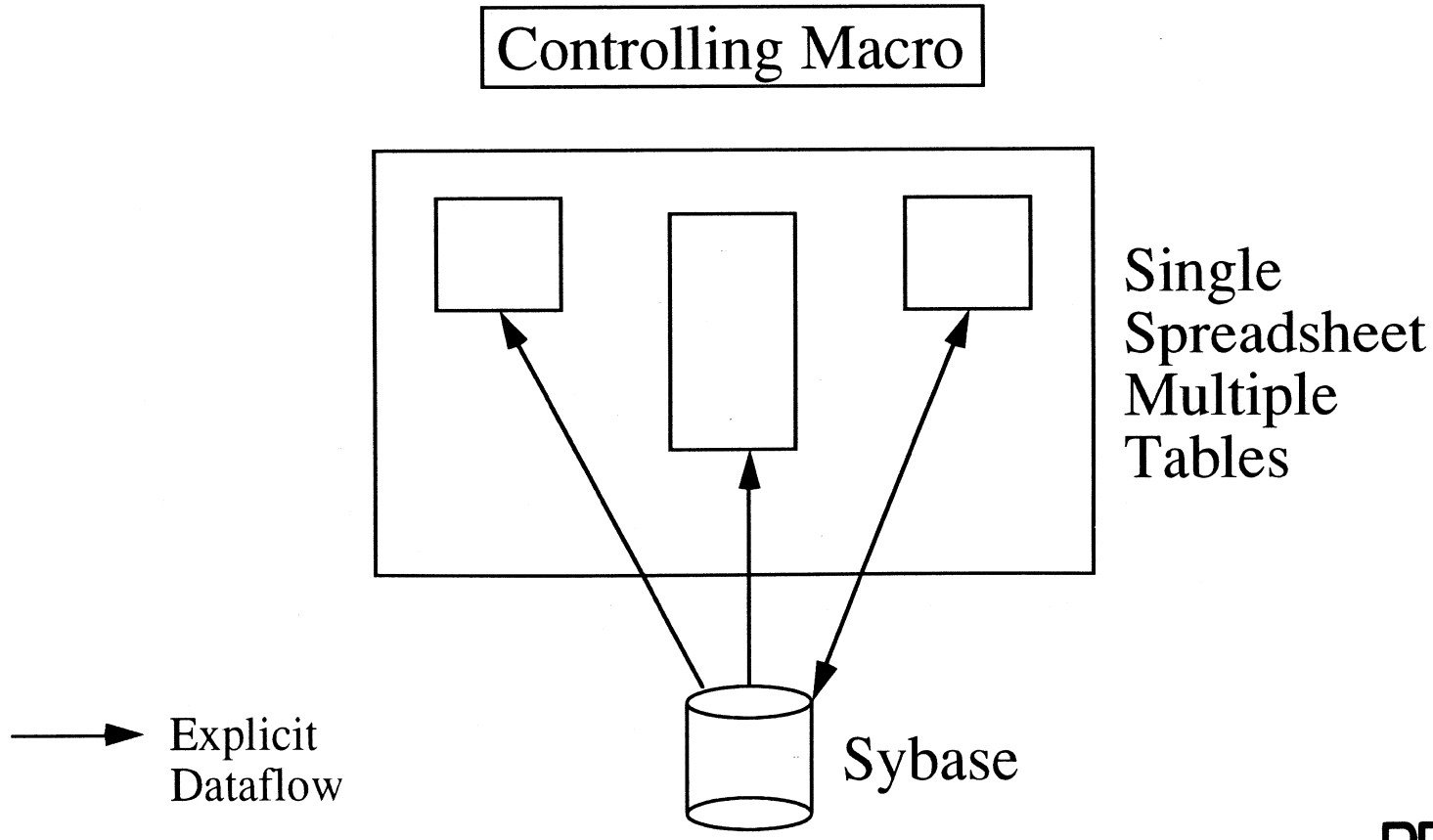
Problems Remaining Or New

- Data Duplication
- Still Too Much Data Movement
- Explicit Sequencing Of Calculations
- Inappropriate Product Use

All problems were addressed by the final design considered by Praxis

PRAXIS

Final Design : Single Sheet Approach



PRAXIS

Final Design

Presentation	Excel
Calculation	Excel
Data Placement	Sequelink
Data Extraction	Sybase
Data Storage	Sybase

PRAXIS

Problems Addressed

- ❑ ✓ Reduced Data Duplication
- ❑ ✓ Very Little Data Movement
- ❑ ✓ Excel Allowed To Sequence Calculations

PRAXIS

Problems Remaining or New

- Single Spreadsheet Limits Data Volume
- Locating Particular Calculations Harder

PRAXIS

Our Original Assumptions

- Logical Data Model Is Spreadsheet Design ?
- LDM Models Object Relationships
- Calculation Relationships Could Be Modelled ?
- Inclusion Of Calculated Attributes

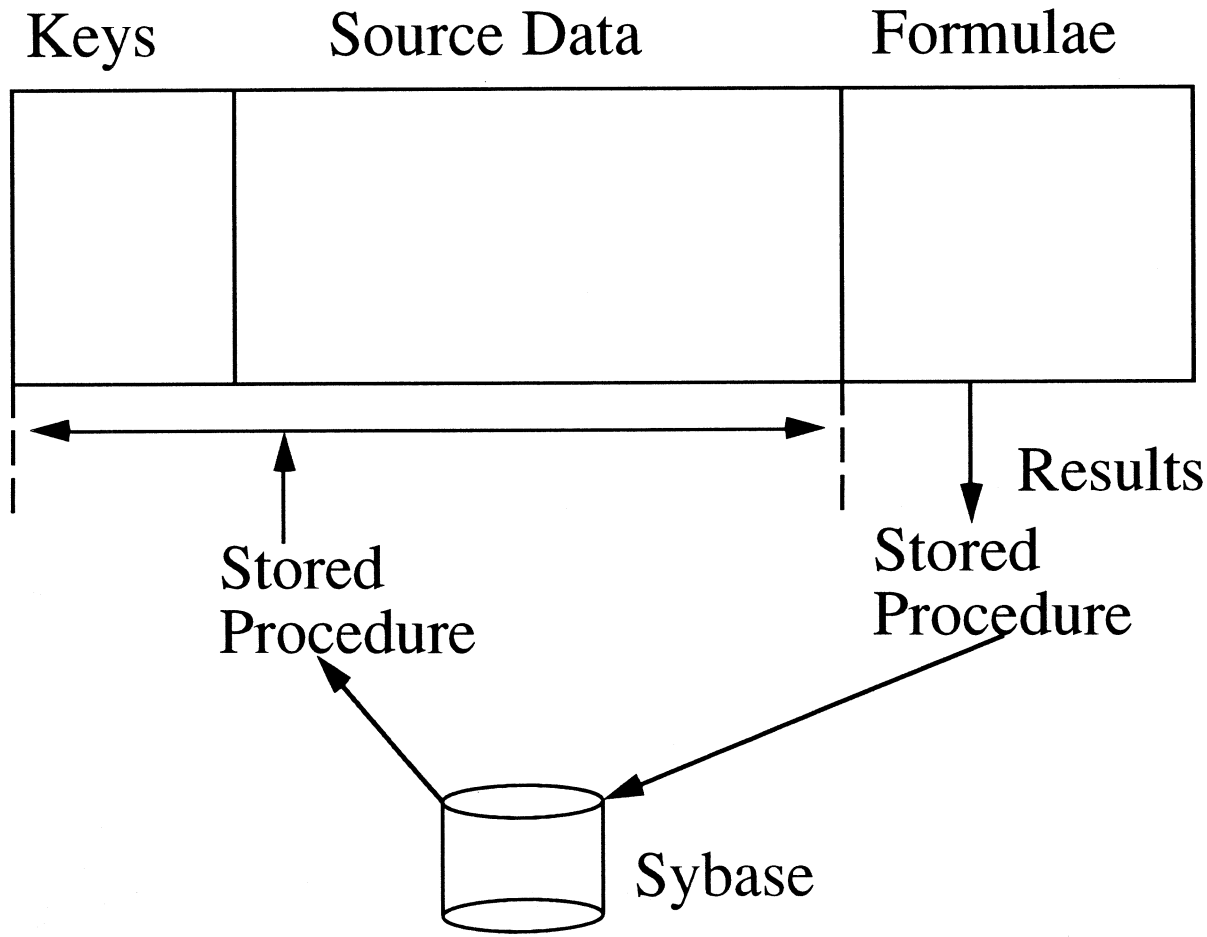
PRAXIS

Okay, But How Do I Design The Table Structures ?

- Many Ways But....
- Specify The Calculations
- Design From The Calculation Specifications

PRAXIS

Spreadsheet Detail



PRAXIS

Calculation Specifications

- Specify Calculations in Very Small Modules
- Specify Source Attributes and LDM tables
- Include Time Periods in Specification
- Order Calculation Specifications

PRAXIS

Design From Calculation Specification

For Each Calculation In Turn

- Place Source Data in Tables With Common Keys and Time Periods
- Use Attribute Names and Table Names From LDM
- Place Formula Into Table With Appropriate Key

End For Each

PRAXIS

Result Of Design Process

- Mechanical Design Process
- Naming Conventions Needed
- Formula Work Left To Right
- Formula Often In Same Table As Data, Not Obscured
- Spreadsheet Resembles LDM
- Spreadsheet Can Be Modified Manually Afterwards

PRAXIS

Sybase !
Three Steps To Easy Interaction

- Seperation Of Responsibility
- Spreadsheet and Table Structure
- Use Of Stored Procedures

PRAXIS

Step 1,
Seperation Of Responsibility

- Right Tool For Right Job
- Location Of Faults Easier

PRAXIS

Right Tools for Right Job

Design	Traditional	1st	2nd	Final
Presentation	Excel	Excel	Excel	Excel
Calculation	Excel	Excel	Excel	Excel
Data Placement	N/A	Excel	Excel & SequeLink	SequeLink
Data Extraction	N/A	Excel SequeLink & Sybase	Sybase	Sybase
Data Storage	N/A	Sybase	Sybase	Sybase

PRAXIS

Step 2,
Spreadsheet and Table Structure

- Match SequeLink Data Format
- Results Are Ready To Write Back
- LDM Names Link Logical Model To Database And To Spreadsheet

PRAXIS

Step 3,
Use Of Stored Procedures

- Avoids Lots Of Text Manipulation
- Reduce Network Traffic
- Pre-Compilation
- Possible Problem With Pre-Compilation
- Allows Complex Logic and SQL
- 1:1 With Spreadsheet Tables For Tracability

PRAXIS

Conclusion

- Get Product Layers Right
- Logical Data Model Is Important But Not Enough
- Plan Your Design

PRAXIS

Stop Press

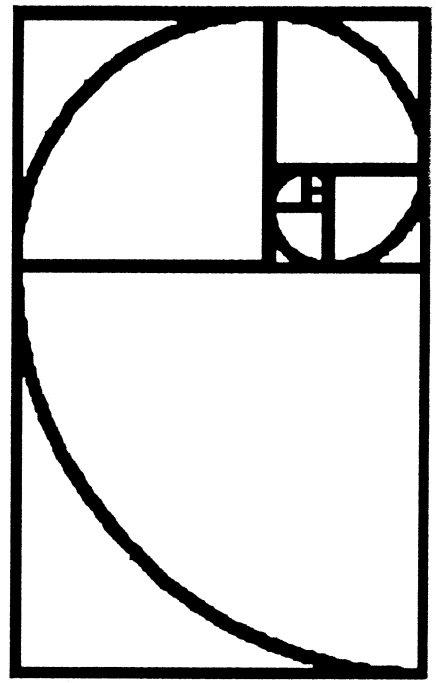
- FORPPAS into System Testing
- Tracability Of Design Demonstrated
- Fault Location Shown To Be Easy
- Design Approach Adopted By 2nd Project

PRAXIS

Developers

Track

Presentations



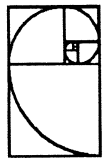
SYBASE

SQL Server Administration

By Jeff Klofft

Introduction

- Controlling SQL Servers
- Monitor SQL Servers

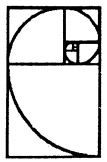


SYBASE

Control Issues

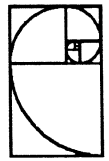
- Small Installation Grows
- Servers Spread Across Wide Area
- Mission Critical Operations
- Time Pressures
- Limited Training

Result: Decreased System Availability



SA Companion Features

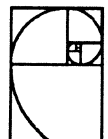
- Intuitive , Window-driven User Interface
- Automates Error-prone Tasks
- Controls an Evolving, Complex Enterprise
- Lets SA Concentrate on What and not How



SYBASE

SA Companion Features

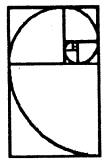
- Server Management
- Device Management
- Database Management
- Server Logon Management
- Reporting



SYBASE

SA Companion Version 10

- Backup Server Support
- Threshold Management Support
- Chargeback Accounting
- New Database Options
- New Configuration Options
- New Roles
- New User Account Features
- New Datatypes



SYBASE

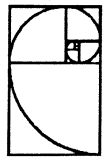
SA Companion Availability

SA Companion Version 1.5.4

Now	Sun/SunOS
	HP9000/HPUX
	RS/6000/AIX
	VAX/VMS

SA Companion Version 10

10/93	Sun/SunOS
	Sun/Solaris
	HP9000/HP-UX
	RS/6000/AIX
	NCR/SVR4

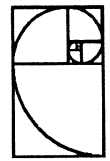


SYBASE

Performance Issues

- Is My SQL Server Configured Properly?
- Are My Database Objects Distributed for Optimum Performance?
- Where is My Processing Burden?
- Is There Any Resource Contention?

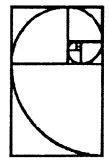
Result: Difficult, if not Impossible, to Configure and Tune SQL Server for Optimum Performance



SYBASE

SQL Monitor Features

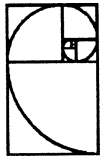
- Easy to Use Graphical Display
- Monitor any Local or Remote Server
- Monitor SQL Server Without Processing Burden



SYBASE

SQL Monitor Features

- SQL Server Statistics
  - Overall Server Performance Indicators
  - General Process Performance Indicators
  - Detailed Process Performance Information

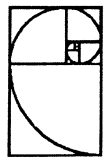


SYBASE

SQL Monitor Features

SQL Server Statistics

- Overall Server Performance Indicators
  - Memory Allocation
  - Performance Summary
  - Performance Trends
  - Transaction Activity
  - Cache Performance
  - Device I/O



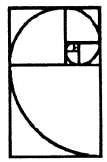
SYBASE

SQL Monitor Features

SQL Server Statistics

- General Process Performance Indicators
    - Process Activity
    - Process List

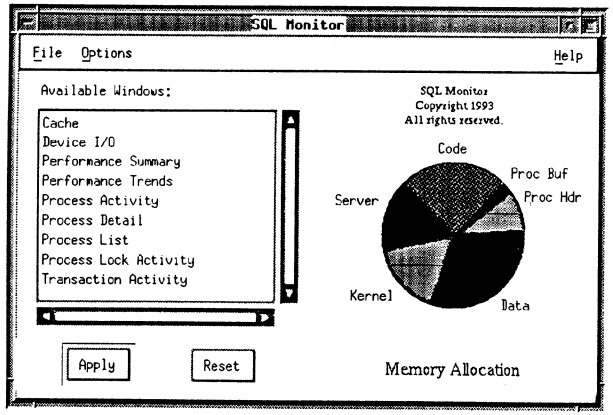
- Detailed Process Performance Information
    - Process Detail
    - Process Lock Activity

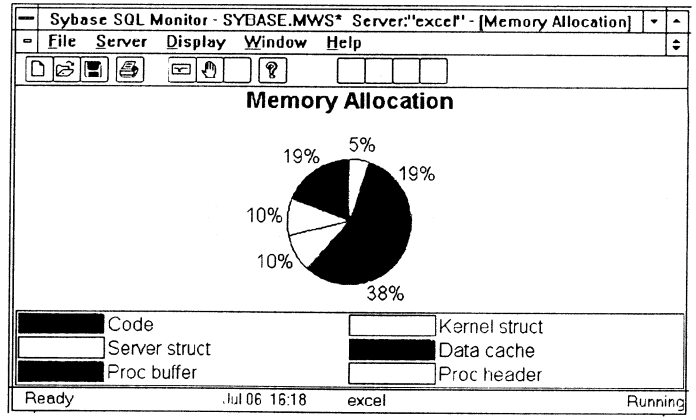


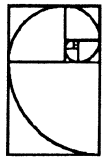
SYBASE

SQL Monitor Features

Memory Allocation







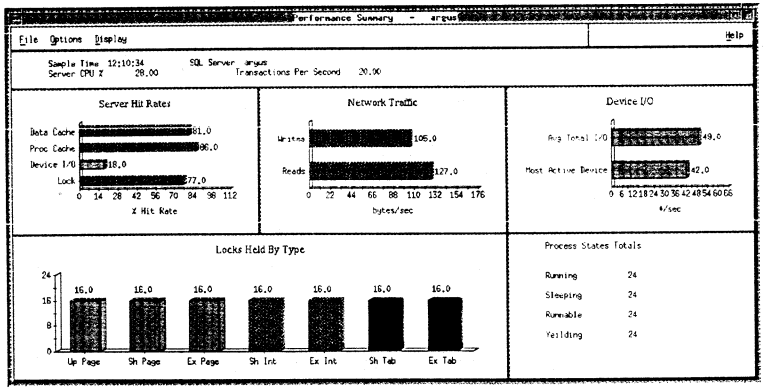
SYBASE

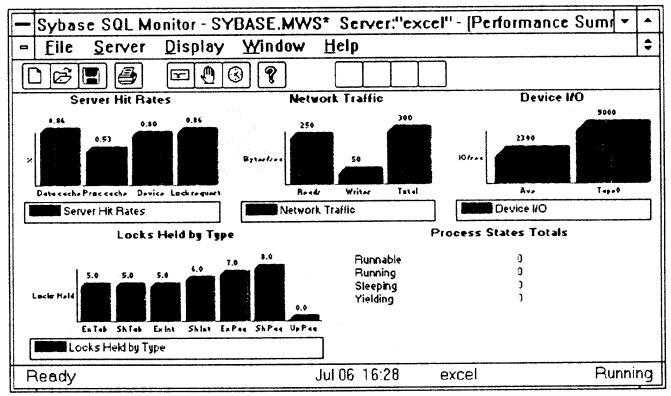
Systems Management Products

Sybase Confidential

SQL Monitor Features

Performance Summary





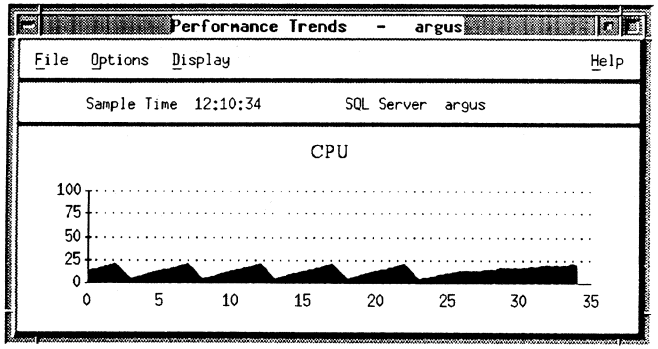
SYBASE

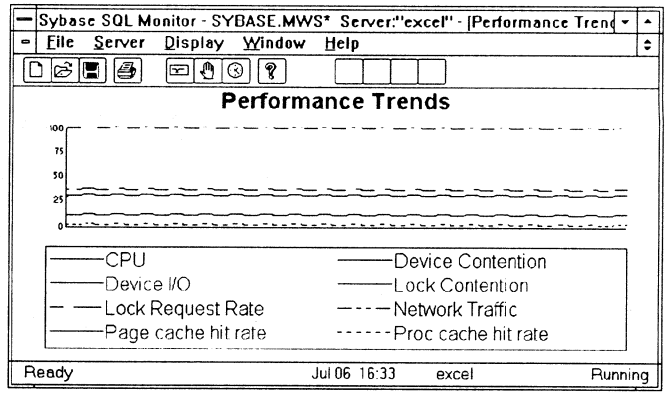
Systems Management Products

Sybase Confidential

SQL Monitor Features

Performance Trends





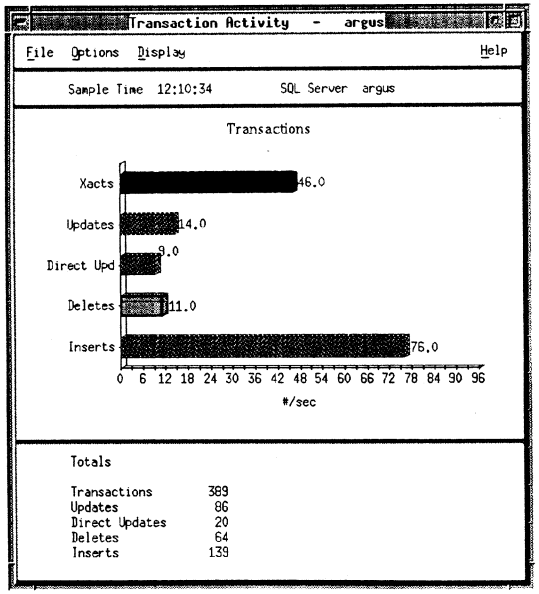
SYBASE

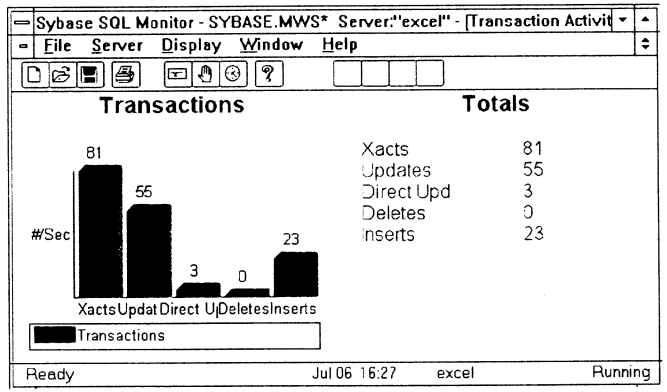
Systems Management Products

Sybase Confidential

SQL Monitor Features

Transaction Activity





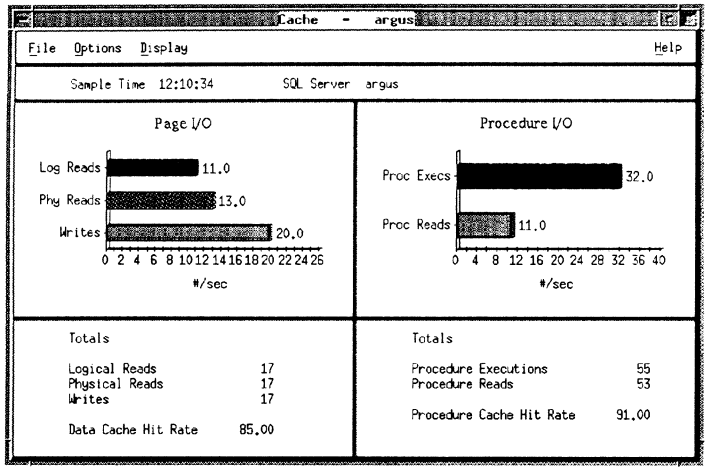
SYBASE

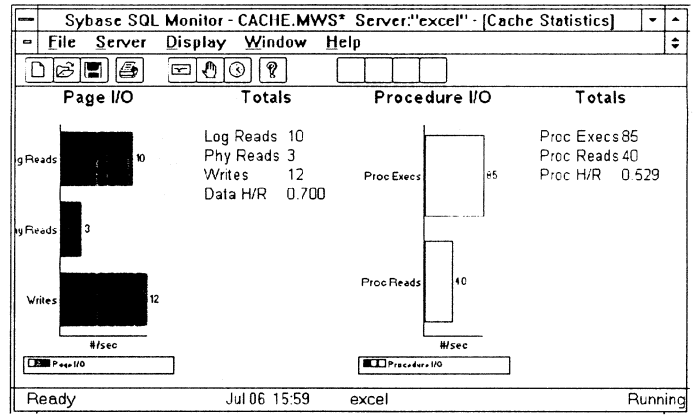
Systems Management Products

Sybase Confidential

SQL Monitor Features

Cache Performance





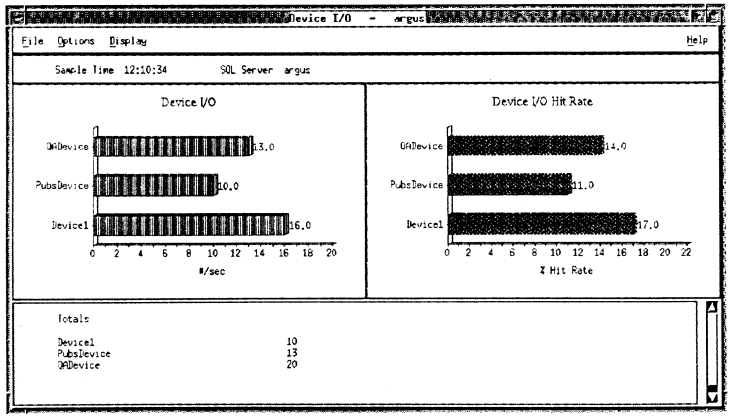
SYBASE

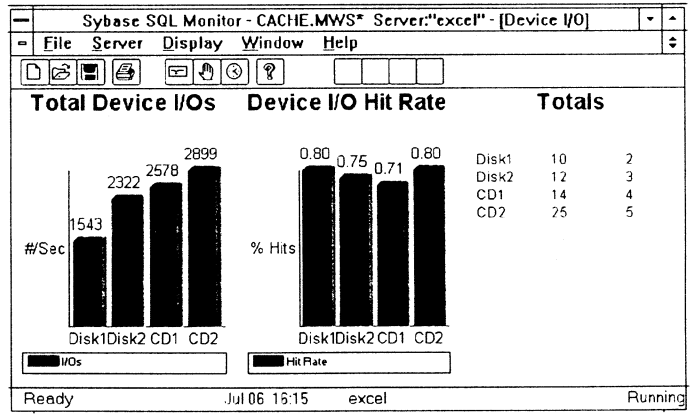
Systems Management Products

Sybase Confidential

SQL Monitor Features

Device I/O - Summary





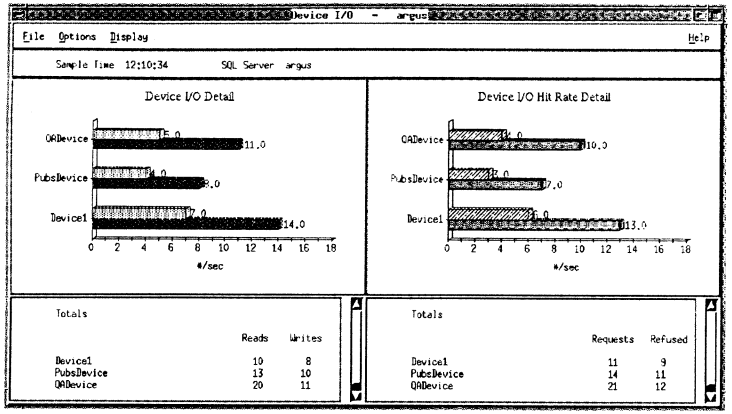
SYBASE

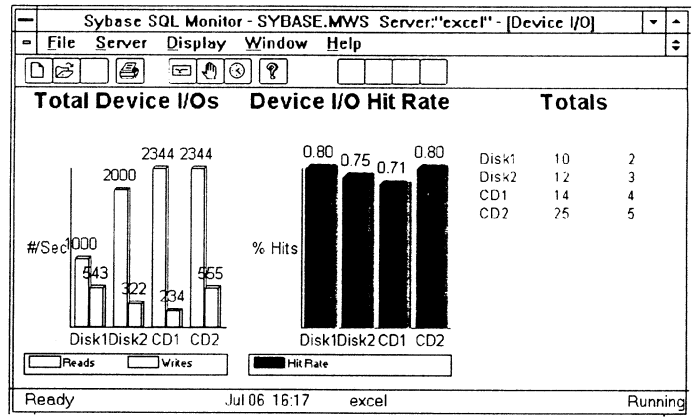
Systems Management Products

Sybase Confidential

SQL Monitor Features

Device I/O - Detail





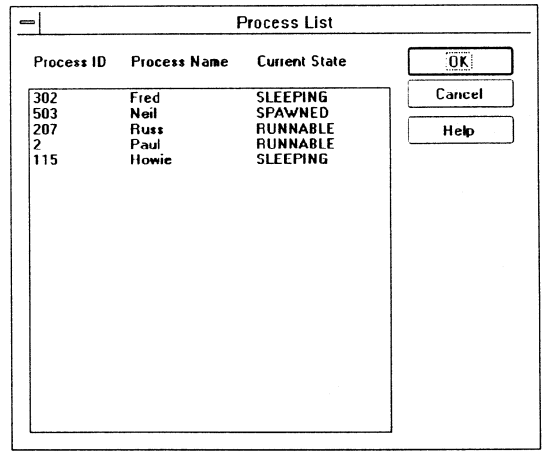
SYBASE

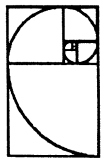
Systems Management Products

Sybase Confidential

SQL Monitor Features

Process List





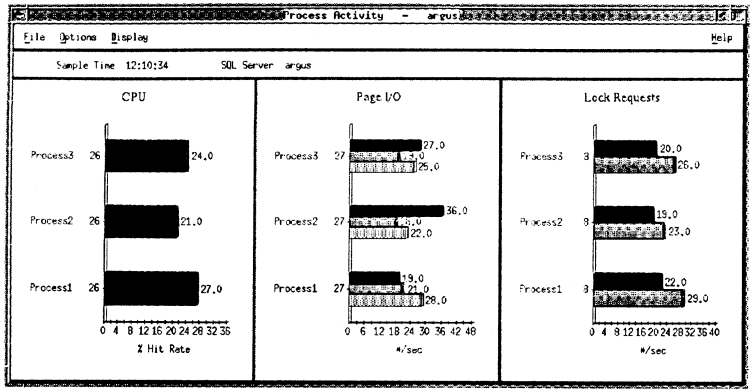
SYBASE

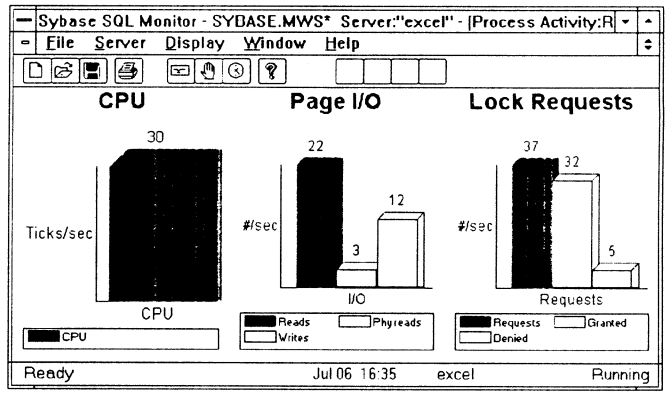
Systems Management Products

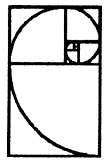
Sybase Confidential

SQL Monitor Features

Process Activity







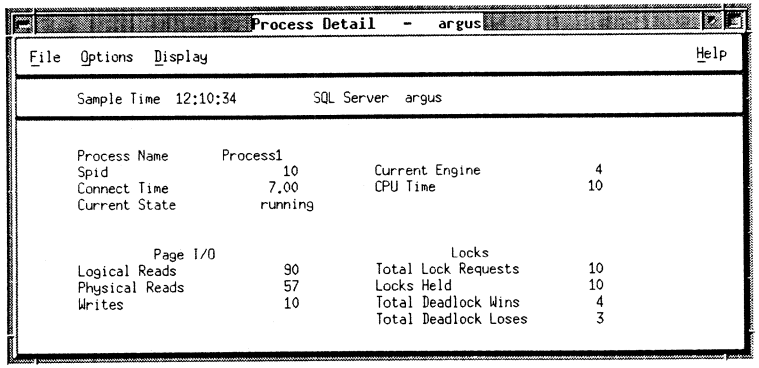
SYBASE

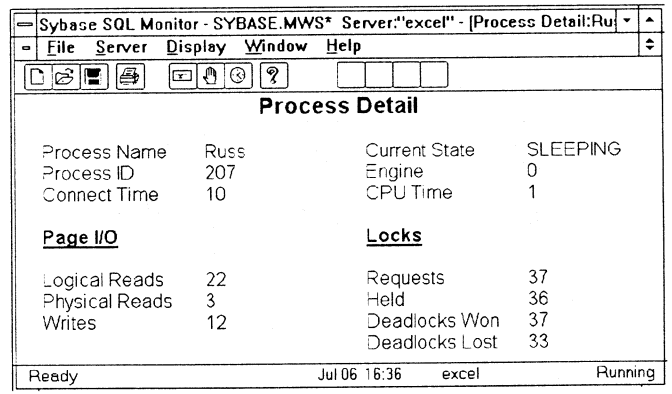
Systems Management Products

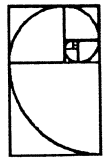
Sybase Confidential

SQL Monitor Features

Process Detail







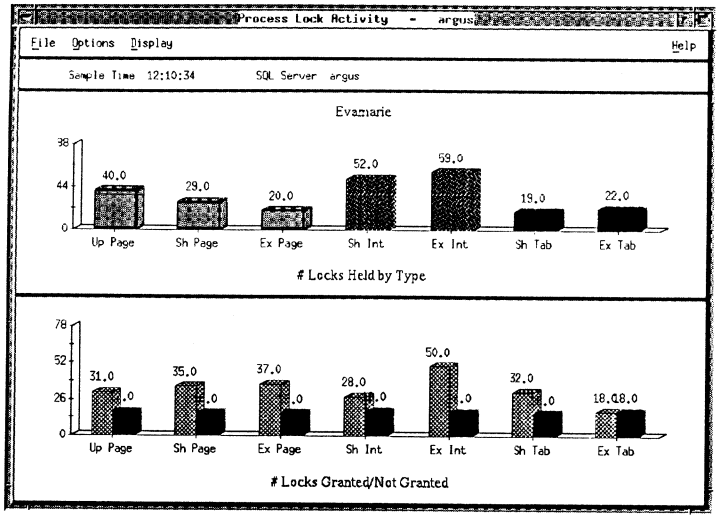
SYBASE

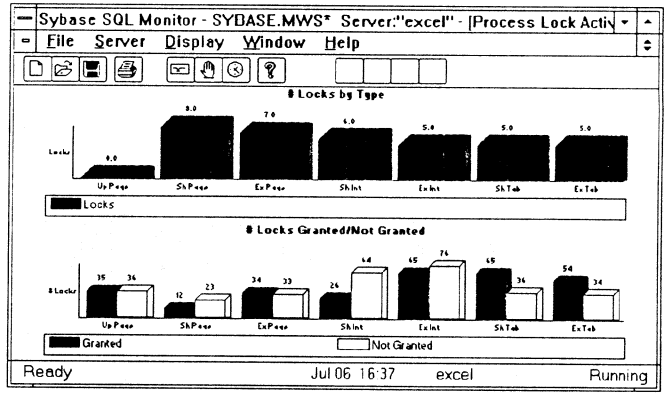
Systems Management Products

Sybase Confidential

SQL Monitor Features

Process Lock Activity - Summary





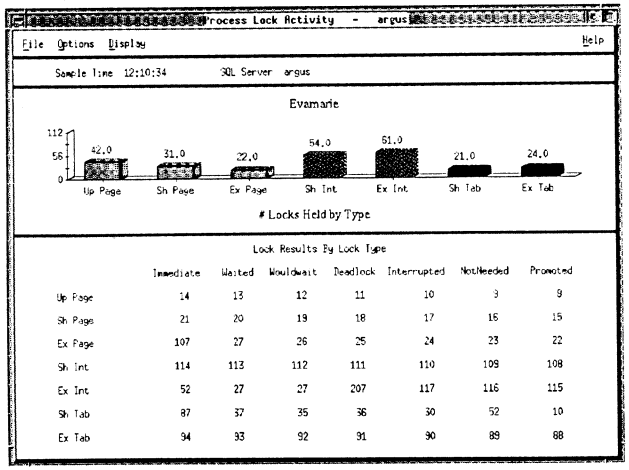
SYBASE

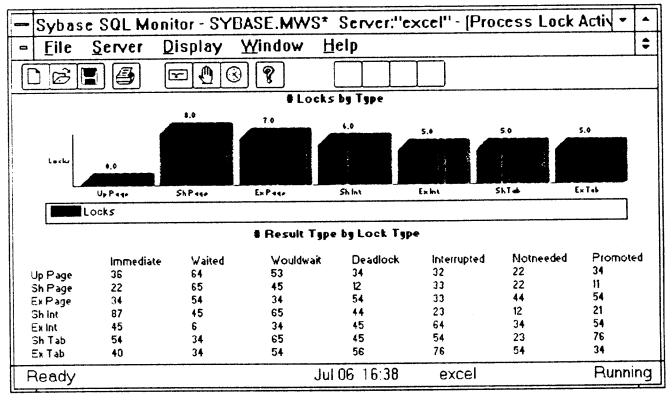
Systems Management Products

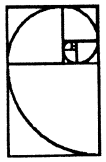
Sybase Confidential

SQL Monitor Features

Process Lock Activity - Detail







SYBASE

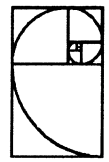
Systems Management Products

Sybase Confidential

SQL Monitor Features

SQL Monitor User Interface Features

- Motif or MS Window 3 Graphical Displays
- Pause and Continue
- Print a Screen
- Print All Screens
- Setting Sample Interval
- Setting Thresholds
- Saving the Display Configuration
- Record and Playback
- On-Line Help
- DDE on MS Windows



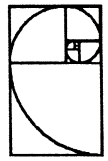
SYBASE

Using SQL Monitor

Most Processes Running Slowly

- Transaction Activity
    - Confirms Below Normal SQL Server Performance

- Performance Summary
    - CPU Busy?
    - I/O Bound?
    - Cache Effectiveness Acceptable?



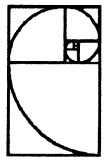
SYBASE

Using SQL Monitor

Most Processes Running Slowly

- Cache Performances
    - Data Cache Hit Rate is 15%
    - Procedure Cache Hit Rate is 90%

- Solution
    - Increase Total Size of the Cache
    - Shift Memory from Procedure to Data Cache
    - Do Both



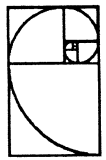
SYBASE

Using SQL Monitor

Some Processes Running Slowly

- Performance Summary
    - CPU Busy?
    - I/O Bound?
    - Cache Performance?
    - Many Locks Held?

- Process Detail
    - CPU Busy?
    - I/O Bound?
    - Lock Status?



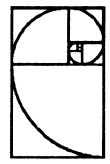
SYBASE

Using SQL Monitor

Some Processes Running Slowly

- Process Lock Activity
    - Exclusive Intent High
    - Exclusive Page High

- Solutions
    - Reduce the Size of the Updates or Inserts
    - Re-Write Queries
    - Change Logical Model

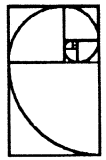


Using SQL Monitor

Most Processes Slower Sometimes

- Performance Trends
    - CPU Busy?
    - Device I/O?
    - Cache Effective?
    - Device Contention?

- Device I/O (Summary Mode)
    - General_Device_1 has many Device I/O Requests
    - General_Device_1 has many I/Os Refused

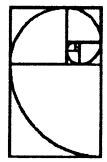


SYBASE

Using SQL Monitor

Most Processes Slower Sometimes

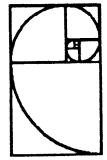
- Solutions
  - Reduce Overall System Load
  - Move General_Device_1
  - Remove Active Databases off of General_Device_1



SYBASE

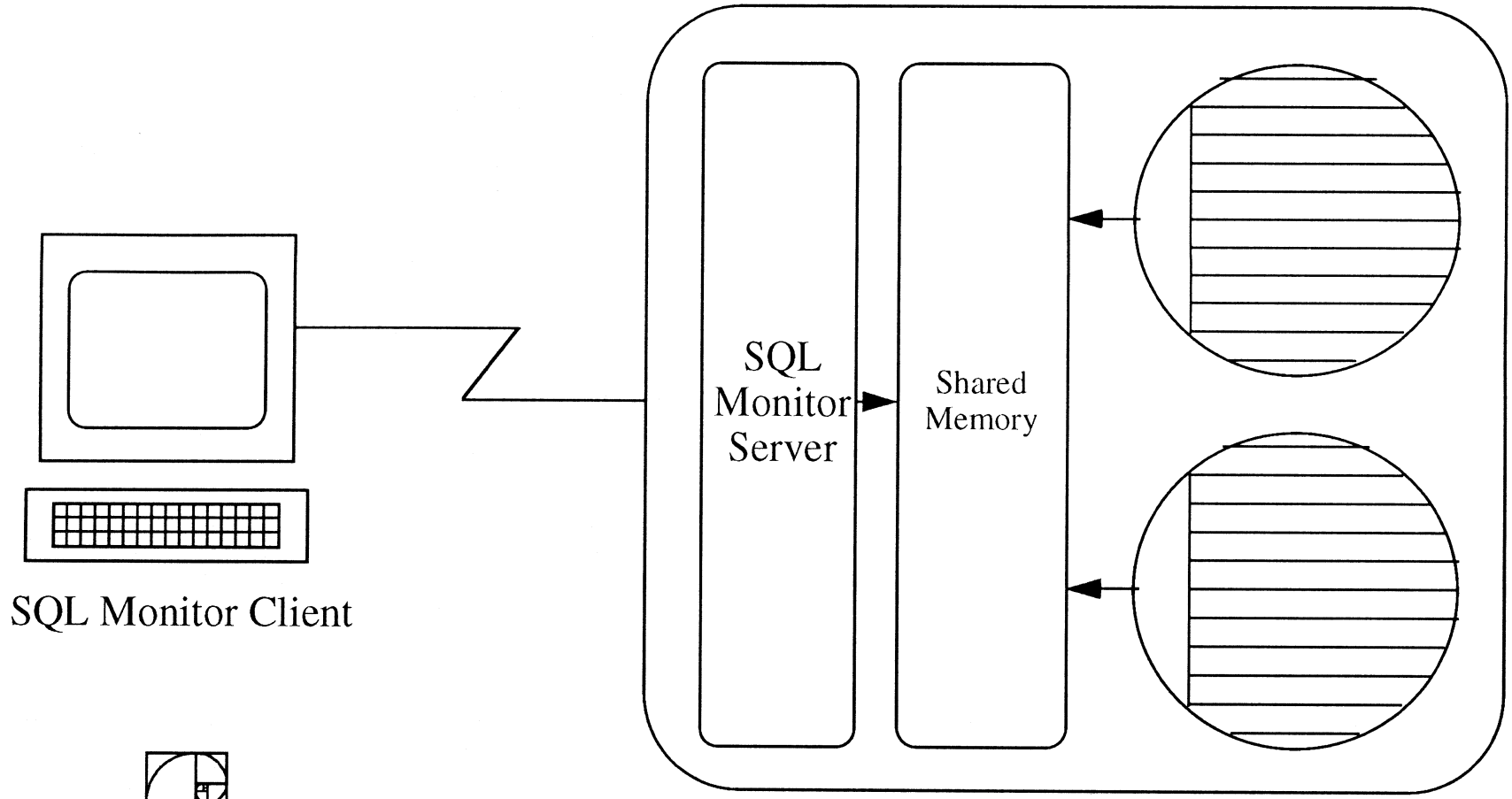
Using SQL Monitor

- Summary
  - SQL Server Performance Problem
  - Application Related Problem
  - System Related Performance Problem



SYBASE

SQL Monitor Internal Architecture



SYBASE

Systems Management Products

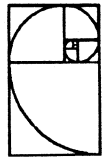
Availability

10.0 SQL Monitor Client

3Q93	Sun/SunOS
4Q93	HP9000/HPUX
	RS/6000/AIX
	NCR/SVR4
	Sun/Solaris

10.1 SQL Monitor Client

3Q93	PC/Windows 3.1
1H94	Unix Platforms

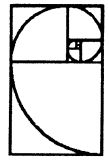


SYBASE

Availability

4.9.1 SQL Monitor Server
3Q93 Sun/SunOS
HP9000/HPUX
RS/6000/AIX
VAX/VMS
NCR/SVR4

10.0 SQL Monitor Server
With System 10 SQL Server



SYBASE

Navigation Server Product Overview

Navigation Server™
Product Overview

S Y B A S E®

Client/Server Architecture for the On-Line Enterprise

Navigation Server™

High Performance Database Management
for
High Capacity Database Requirements

Developed by NCR and Sybase

Major Features

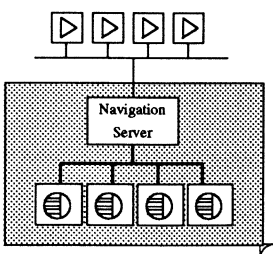
- Scalable Performance
- Transparent Access
- High Availability
- Leverages Parallel Computing Platforms
- Graphical Administration Interfaces
- Platform Independence

Performance

- Parallel Access to Partitioned Data
- Scaleup
  More transactions on greater database size
- Speedup
  Greatly reduced response time on large tables

Transparent Access

- One Logical Schema
  Application Programmer
  End User
- Appears as Single SQL Server
- Same Sybase Interfaces
- Interoperates with Sybase System 10 Products



Target Environments

- OLTP beyond 1000 tps
- DSS (Analytic Processing)
- DSS (Information Retrieval)
- Batch Processing
- OLCP (On-Line Complex Processing)

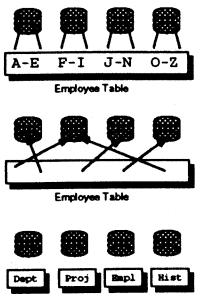
Very Large Database
Single Platform

Navigation Server Product Overview

Data Partioning and Parallelism

- Horizontal Partitioning of Tables
- Partitioning Choices
- Global Directory
- Partitions Assigned to SQL Servers
- Translation of SQL to Parallel SQL
- Parallel Access

Partitioning Choices



Partitioning Choices

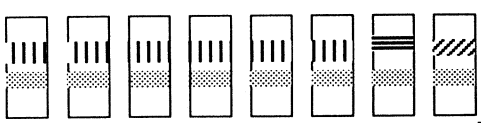
- Range
  - Value Ordered Requests
  - TPC/A or TPC/B type transactions
- Hash
  - Even Workload
  - Access to Entire Table
- Schema
  - Small Tables Not Partitioned

Future Partitioning Choices

- Replicated Small Tables
  - Join Performance
- Vertical (column sets)
  - Mixed Data Types - Blobs, Images
  - Physically Distributed Partitions
- Time Related
  - Mixed Data Storage Media

Partitions - Homes - Nodes

- Table Partitioned Across All or Subset of Nodes
- Collection of Nodes for Table = Home



Defining Partitions

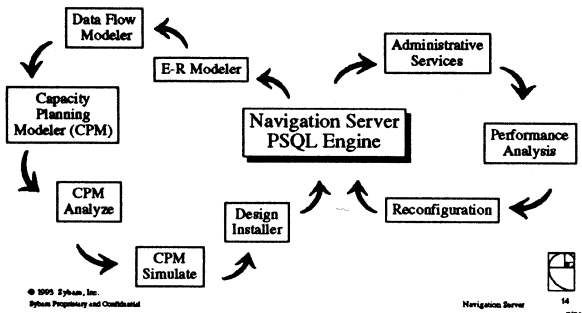
- Navigation Server Controls the Placement
- Calculated
  - via Configurator™
- Default
  - default scheme in the Global Directory

Navigation Server Product Overview

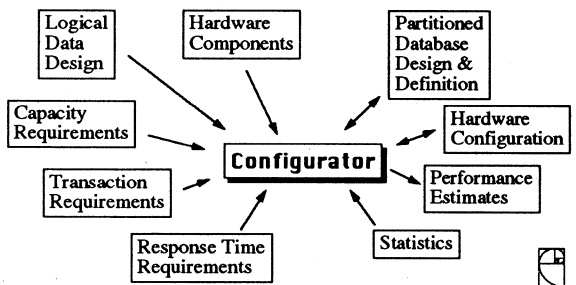
Global Directory

- Private Database
- Repository Necessary to Run the System
- Configuration Data
- Partitioning Information
- Access Via Navigation Server Administrative Services (NSAS)
- Controlled by Schema Server
- Always Mirrored

Navigation Server Lifecycle



The Configurator™

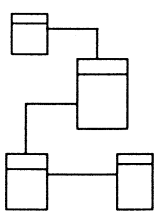


Configurator™ Uses

- Response to RFPs
  - Configurations
  - Performance Simulation
- Applications & Database Definition
- System & Database Installation
- System Tuning & Reconfiguration
- System Upgrades
- Capacity Planning
- Effects of Upgrade $$

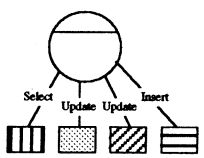
Entity Relationship Modeling (ERM)

- Logical Data Modeling
  - Tables - Rows - Indexes
- Table Capacity Estimates
- Based on Deft Tool
- GUI or DDL



Data Flow Modeling (DFM)

- Define Transactions
- Define Queries
- GUI or Transact SQL
- Extensions
  - # of Sources
  - Throughput Requirements
  - Response Time Requirements
  - Rates per Intervals



Navigation Server Product Overview

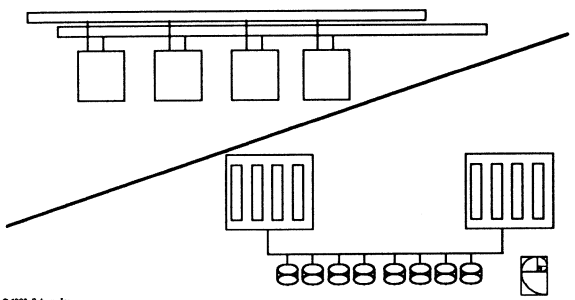
Capacity Planning Modeling (CPM)

- Generates hardware, software, and partitioning based upon ERM and DFM
- Analyzes model of the workload(s)
- Simulates the workload(s)
- Distribute - determine new partitioning based on existing workload
- Performance Reports
- Performance Graphics

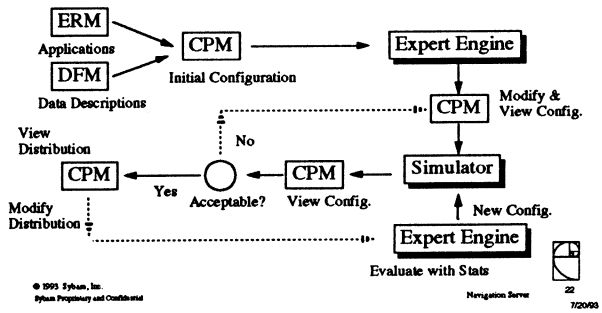
Configuration Generation

- Select Application(s)
- Choose Constraints
  CPU, Disk Capacity, Interconnect, Memory, Mirroring
- Choose Table Growth %
- Choose Transaction Rate Growth %

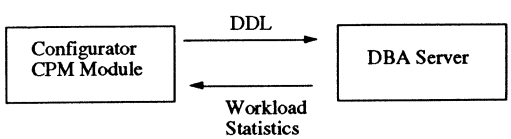
Configuration Output



Configurator Flow



Configurator Integration

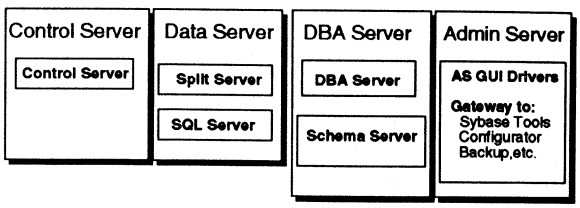


Other Database Defintions

- DBA Can Define "Home"
- No Partitioning Desired
- Quick Alterations Not Affecting Partitioning
- Extended Stored Procedure Facility in Admin. Svcs.

Navigation Server Product Overview

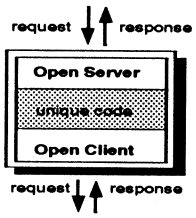
Basic Components



Communicate Via Message Passing: Shared Nothing

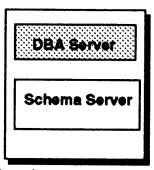
Component Structure

- Taking Advantage of Sybase Open Architecture



DBA Server

- Executes Parallel SQL Compiler
- DDL Interpreter
- Global Directory Functions
- Recovery Management - global deadlock detection
- Mirrored on Alternate Node (LCMP platform)
- Multiple DBA Servers
- Initiates Backup/Restore
- Schema Server Holds Global Directory

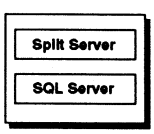


Control Server

- Receives Requests From Clients
- Initiates Parallel SQL Execution
- Sends Parallel Requests to Split Servers and SQL Servers
- Monitors and Initiates Recovery for Data Servers

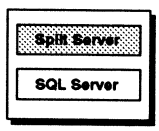
Data Server

- Unit of Parallelism
- Control Server Handles Recovery for Unit
- SQL Server Can Be VSA or Uni



Split Server

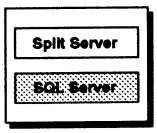
- Redistributes & Replicates Tables for Joins
- Merges Results from Split Tables
- Creates & Destroys Temp. Tables



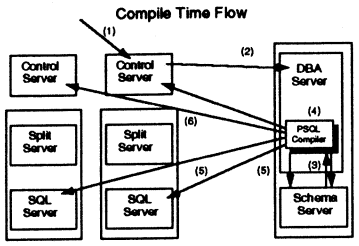
Navigation Server Product Overview

SQL Server

- Executes SQL Component of Parallel Plan
- uni-processor or VSA mode
- Standard SQL Server



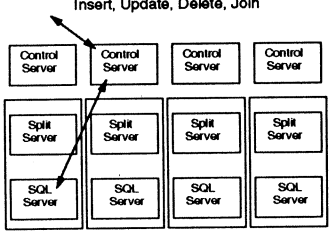
PSQL Compilation



(1) Request Received
(2) Passed to DBA Server
(3) Schema Server for Parsing
(4) Compilation and Optimization
(5) Stored Procedure Sent to All Servers Participating
(6) Loaded Into Memory for Execution

Single Partition Actions

Run Time Flow: Single Partition
Insert, Update, Delete, Join



(1) Request Received
(2) Request sent to single SQL Servers
(3) Results back to Control Server
(4) Results are returned to requesting client

Multiple Partition updates handled with 2-phase commit

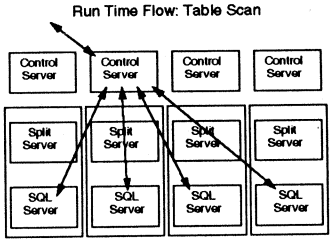
Multiple Control Servers Acting Concurrently

Transaction Processing Scaleup

- Nearly Linear for # of Partitions
- Assuming Transaction Load is Partition Balanced

Table Scan

Run Time Flow: Table Scan



(1) Request Received
(2) Stored Procedure Requests sent to all participating SQL Servers
(3) SQL Servers process stored procedure
(4) Control Server merges results and reorders if needed
(5) Results are returned to requesting client

Table Scan Speedup

- 25% overhead

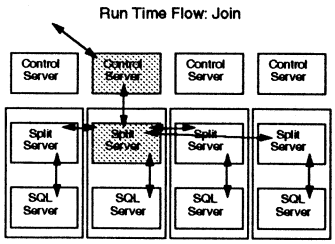
New time = Old time / (# of Data Servers * .75)

Old time of 60 minutes moved to Navigation Server with 16 partitions

60 / (16 * .75) = 5 minutes

Navigation Server Product Overview

Joins Crossing Partitions



(1) Request Received
(2) One Split Server becomes master
(3) Data is restributed as temporary tables
(4) In-place joins
(5) Results are merged
(6) Results are returned to requesting client

Types of Joins

- Redecluster 1
  Move rows from one table into home of other table
- Redecluster 2
  Move rows from both tables into a new home
- Replicate 1
  Replicate one table onto partitions of other tables home

Need Enough Work Space!

Complex Join Speedup

- 40% Overhead

New Time = Old Time / (# of DataServers * .60)
60 minutes with 16 partitions
60 / (16 * .60) = 6.25 minutes

Navigation Server Administrative Services (NSAS)

- Reduces Managing Complexities
  Hides Operating System
- Seven Applications - all GUI
  - Backup/Restore
  - Configuration Management
  - Designs
  - Stored Procedures
  - Logs
  - Tools

Backup/Restore

- Built Upon Sybase Dump/Load
- Single View of Data and Logs
- Parallel Execution
- Flexibility for Partition Restore
- Automatic or Time Specified
- Integrated With Install Design Facility

Greatly Reduced Backup Time!

Configuration Management

- Hardware & Software Configuration
- View of Software Mapped to Hardware
- Peripheral Mapping
- Database Topology
- Alerts & System Notifications
- System Maintenance Facilities

Navigation Server Product Overview

Designs

- Gateway to Configurator™
- Install Design Facility

Stored Procedures

- Similar to ISQL
- Auto Prompting for Parameters
- Results into Separate Windows
- Extensive On-Screen Help
- Accesses Global Directory

Log Management

- System/Navigation Server Statistics
- Error Logs
- Trace Logs
- Log to Unix files
- Display to Operator

Tools

- Gateway to isql
- Gateway to dbcc
  dbcc on individual partitions!
- Gateway to bcp

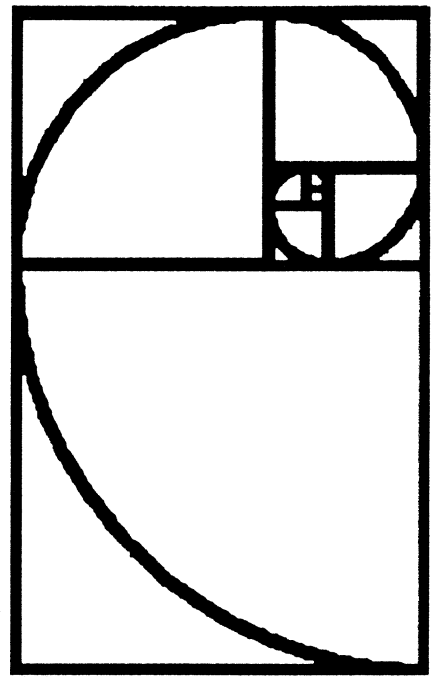
Application Qualification

- Very Large Single Store Database
- Maybe
  Large Number of Users
  Very High Transaction Rates
  Desire for "Open Solution"
  Need for Scalability
  Need for Flexibility

Navigation Server™

High Performance Database Management
for
High Capacity Database Requirements

Developed by NCR and Sybase



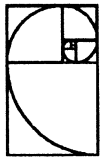
SYBASE

Debugging Transact-SQL

By Jeff Klofft

Introduction

- Developer's Problem
- SQL Debug Features
- Using SQL Debug
- Internal Architecture
- Availability
- Futures

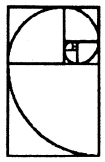


SYBASE

Developer's Problem

- Using 1960's Style Debugging for Transact-SQL
    - Sprinkling in Print Statements
    - Extracting Code Sections and Executing in ISQL
- Cannot Use Existing Debugging Skills
- Losing 3GL Debugging when Using 4GL
- Need to Understand New Code
- Application Split Across Network

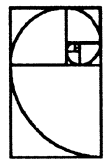
Result: Longer Time to Find and Correct Bugs



SYBASE

SQL Debug Features

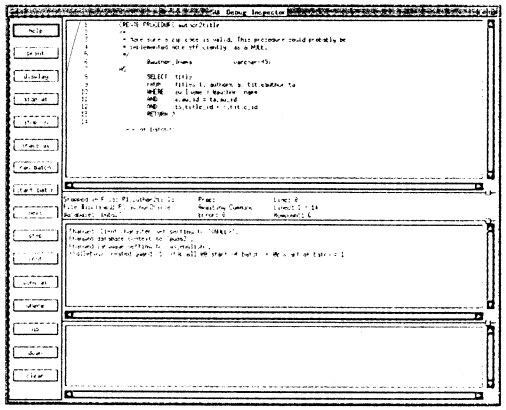
- Graphical User Interface
- Execution Control
- Variable Examination and Control
- Stack Information
- Transparent Debugging
- Multi-Connection Debugging
- Activity Log
- Configurability



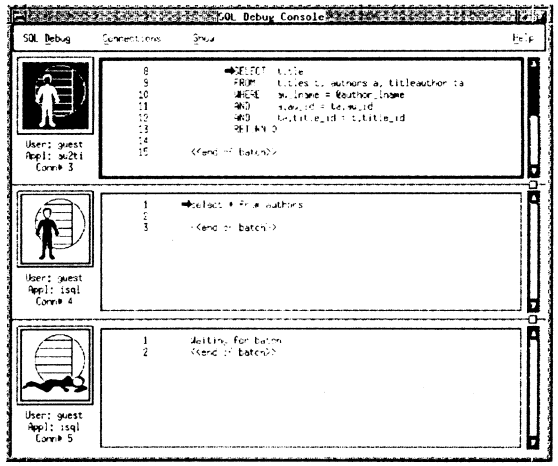
SYBASE

SQL Debug Features

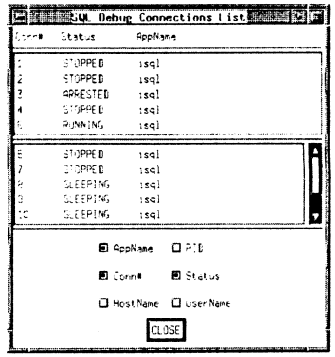
Graphical User Interface



Inspector



Console

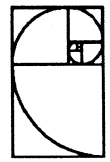


Connections List

SQL Debug Features

Execution Control

- Breakpoints
    - Conditional (Expression Dependent)
    - Unconditional
- Single and Multi-Step Execution
- Dynamic Interruption of Execution Flow
- Variable Execution Entry

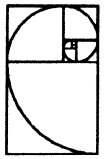


SYBASE

SQL Debug Features

Variable Examination and Control

- Dynamic Assignment of Variables
- Full Global Variable Support
- Print Current Values of Local and Global Variables
- Constant Display of Variable Contents

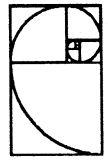


SYBASE

SQL Debug Features

Stack Information

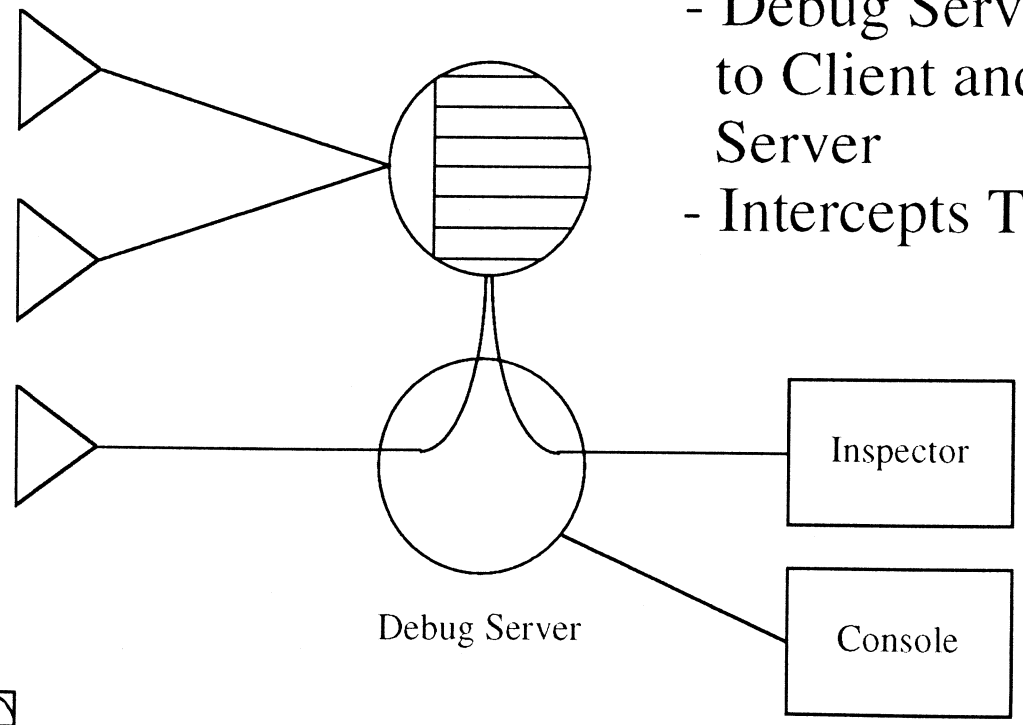
- Stack Contents
- Current Stack Position
- Ability to Move Up or Down the Stack Frames



SYBASE

SQL Debug Features

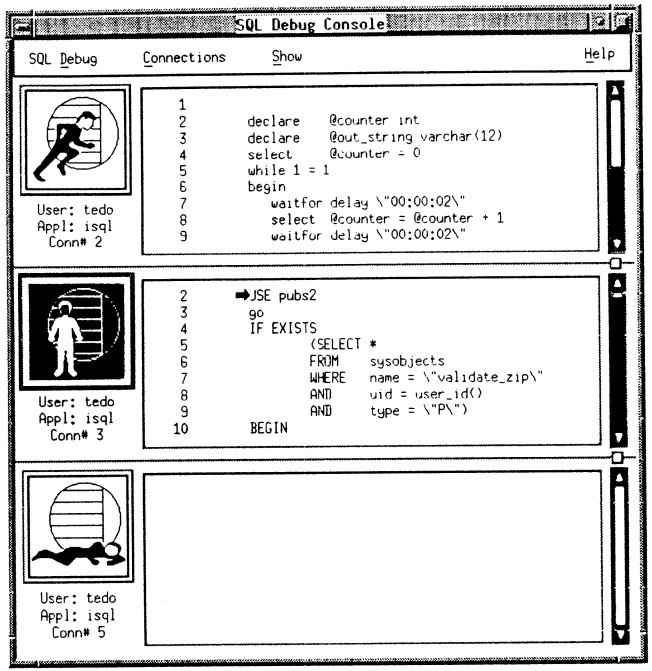
Transparent Client Debugging



- Debug Server is Open Server to Client and Client to SQL Server
- Intercepts Transact-SQL

SQL Debug Features

Transparent Client Debugging



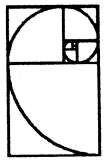

Running


Stopped


Sleeping


Tracing


Arrested



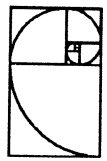
SYBASE

Systems Management Products

SQL Debug Features

Transparent Client Debugging

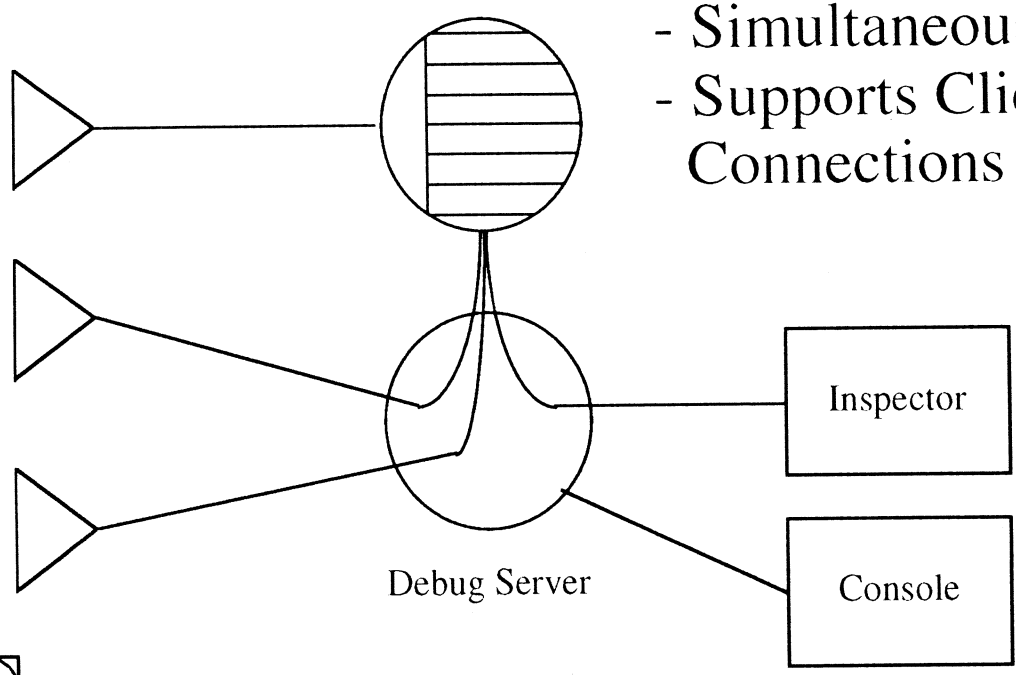
- Debug User Can See Generated SQL
- Supports any Open Client Application
    - APT
    - Gain
    - SQR
    - PowerBuilder
- Examine Rows and Messages Returned to Client



SQL Debug Features

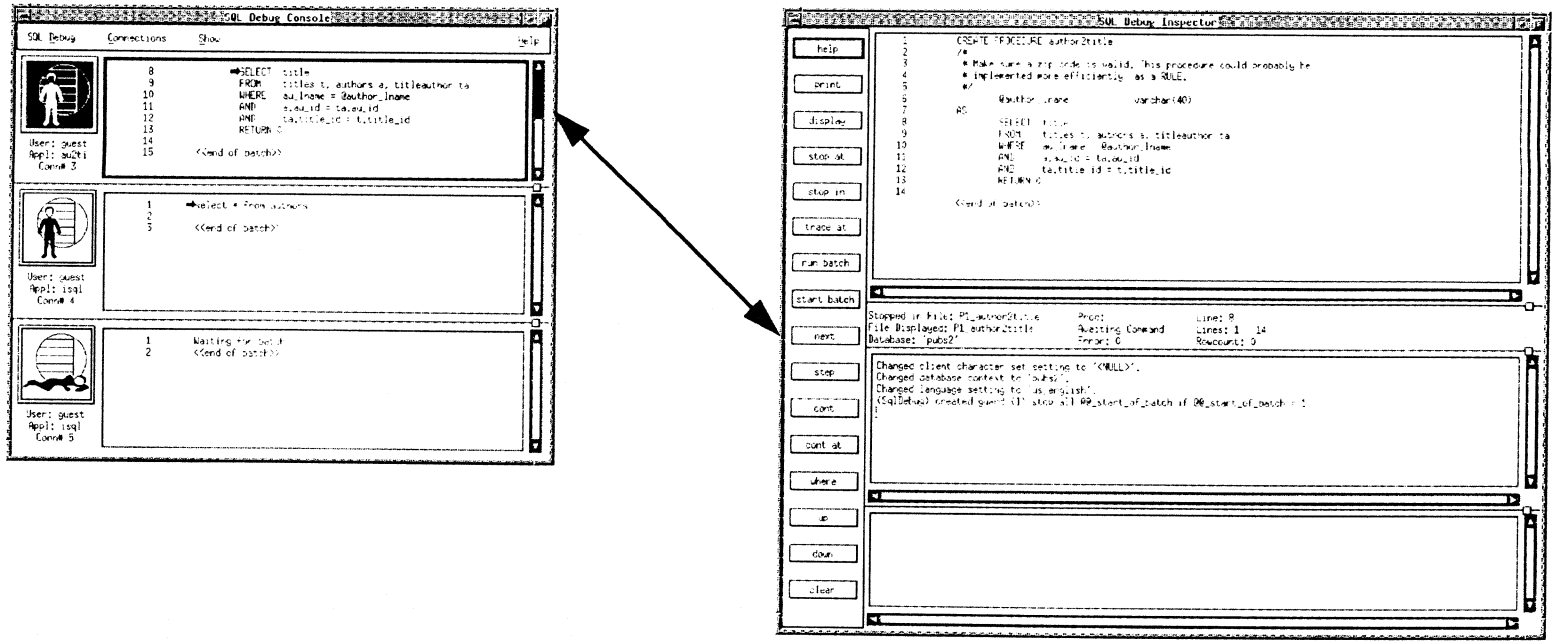
Multi-Connection Debugging

- Simultaneous Viewing
- Simultaneous Manipulation
- Supports Clients with many Connections



SQL Debug Features

Multi-Connection Debugging



SYBASE

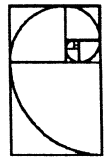
Systems Management Products

Sybase Confidential

SQL Debug Features

Multi-Connection Debugging

- Watch Interaction of Several Connections
- Resolve Deadlock Situations
- Cause Deadlock and Other Conditions to Test Error Handling in Client Application

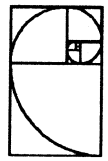


SYBASE

SQL Debug Features

Activity Log

```
Apr 27 12:29:07 1993 1 Received  CONNECT event
Apr 27 12:29:07 1993 1 Username = 'gil'
Apr 27 12:29:07 1993 1 Application = 'isql'
Apr 27 12:29:07 1993 1 Hostname = 'sycamore'
Apr 27 12:29:07 1993 1 Hostid = '28390'
Apr 27 12:29:07 1993 1 Servername = 'SYBASE'
Apr 27 12:29:07 1993 1 CONNECT Succeeded
Apr 27 12:29:07 1993 1 Received LANGUAGE event
Apr 27 12:29:07 1993 1 select *\n
Apr 27 12:29:07 1993 1 from titleauthor \n
```

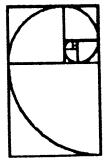


SYBASE

SQL Debug Features

Configurability

- Add, Delete, Change Buttons on Inspector
- Numerous Configuration Variables
- Key Stroke Macros
- X Resources



SYBASE

Using SQL Debug

Transparent Debugging

Name

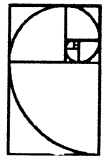
au2ti - display the title written by an author

Synopsis

au2ti [-S server] [-D database] [-U username]
[-P password] -L lastname

Options

-S server	The name of the server containing the database.
-D database	The name of the database containing the titles, authors, and titleauthor tables.
-U username	A valid SQL Server login.
-P password	The password the the username.
-L lastname	The lastname of the author.



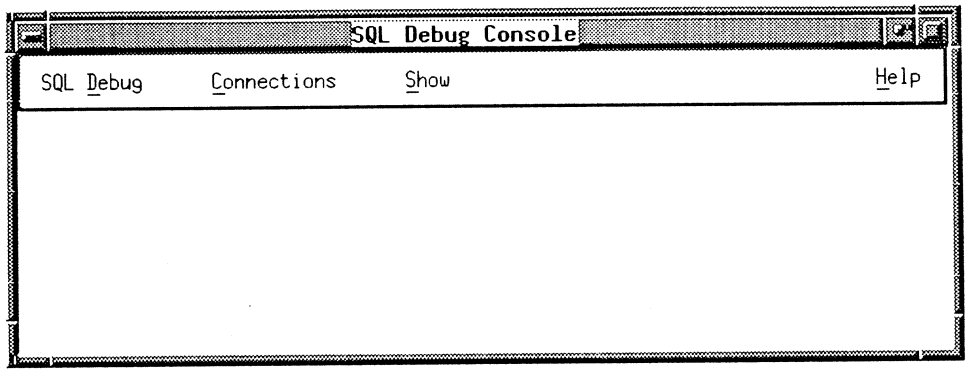
SYBASE

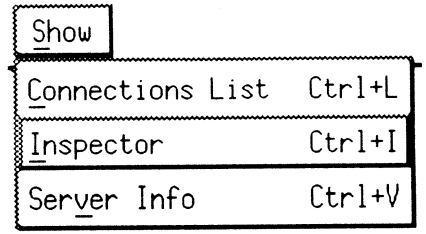
Systems Management Products

Using SQL Debug

Transparent Debugging

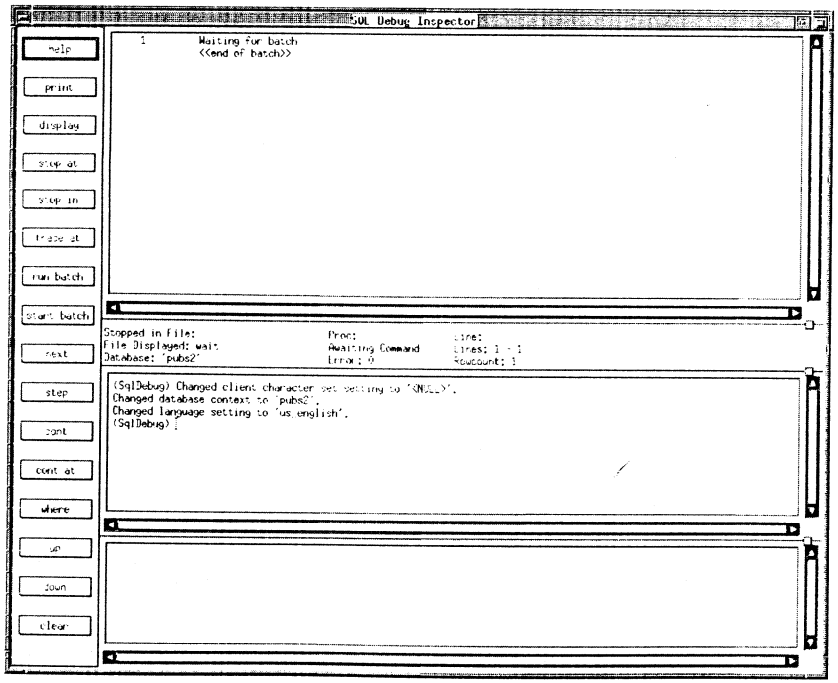
```
% au2ti -L Green
No titles by author Green

% sqldebug -M
```





Using SQL Debug

Transparent Debugging



SYBASE

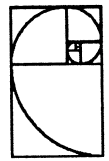
Systems Management Products

Sybase Confidential

Using SQL Debug

Transparent Debugging

```
(SqlDebug) sql
SQL> use pubs2
SQL> go
Changes database context to 'pubs2'.
Execution completed.
SQL>Select title
2> from titles, author, titleauthor
3> where au_lname = "Green"
4> and author.au_id = titleauthor.au_id
5> and titleauthor.title_id = title.title_id
6> go
```

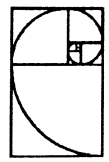


SYBASE

Using SQL Debug

Transparent Debugging

```
Title
-----------------------------------------------------
The Busy Executive's Database Guide
You Can Combat Computer Stress!
Execution complete.
SQL>
SQL> dbg
2> go
(SqlDebug) exit
```

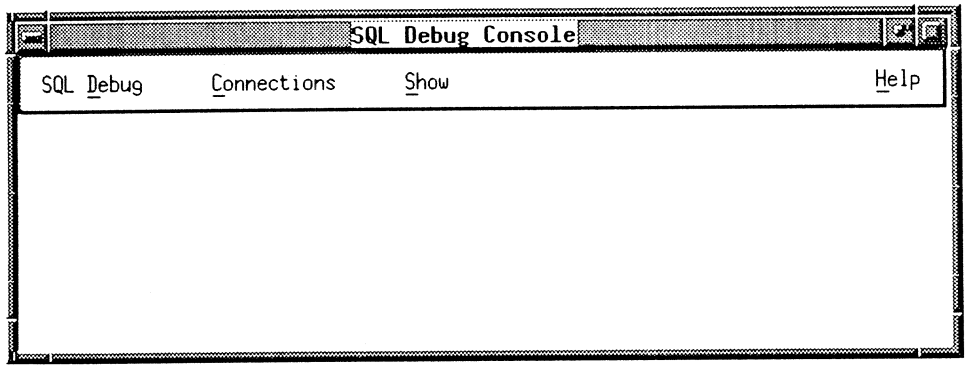


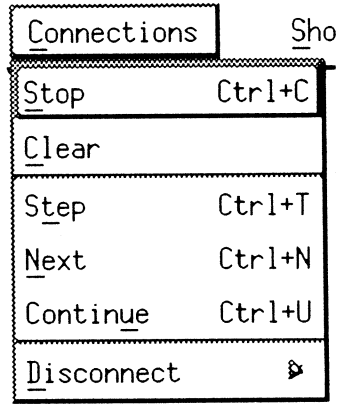
SYBASE

Systems Management Products

Using SQL Debug

Transparent Debugging

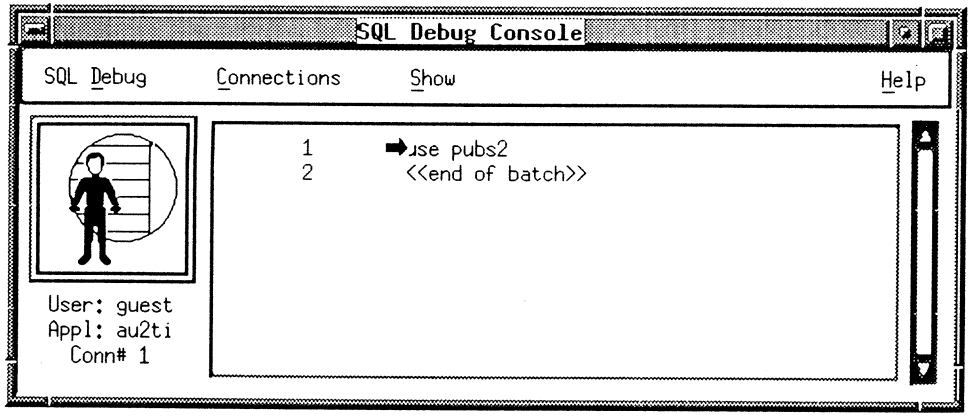





```
% au2ti -S DEBUGGER -L Green
```

Using SQL Debug

Transparent Debugging



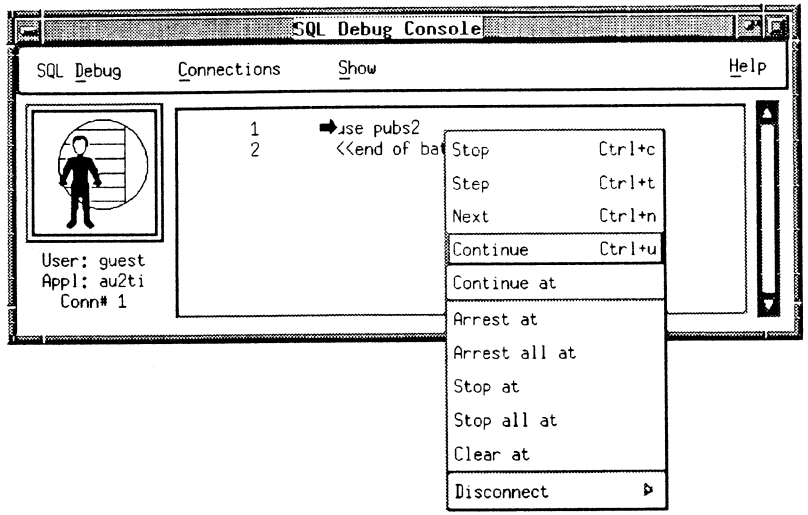
SYBASE

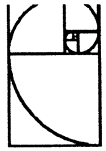
Systems Management Products

Sybase Confidential

Using SQL Debug

Transparent Debugging





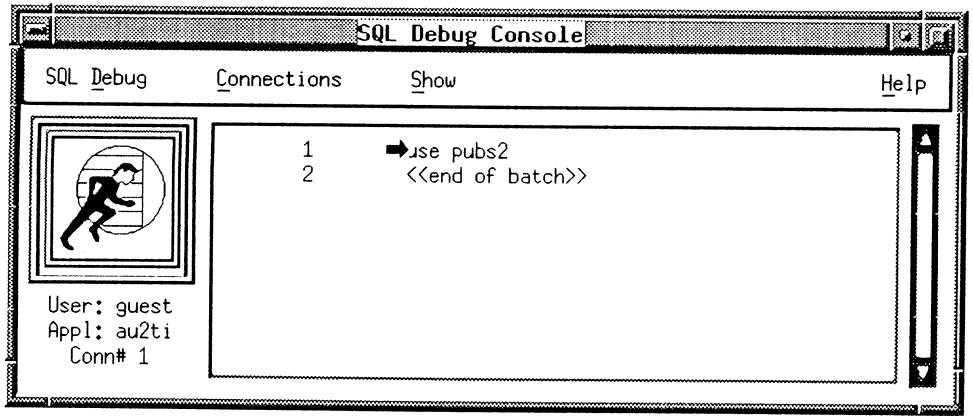
SYBASE

Systems Management Products

Sybase Confidential

Using SQL Debug

Transparent Debugging



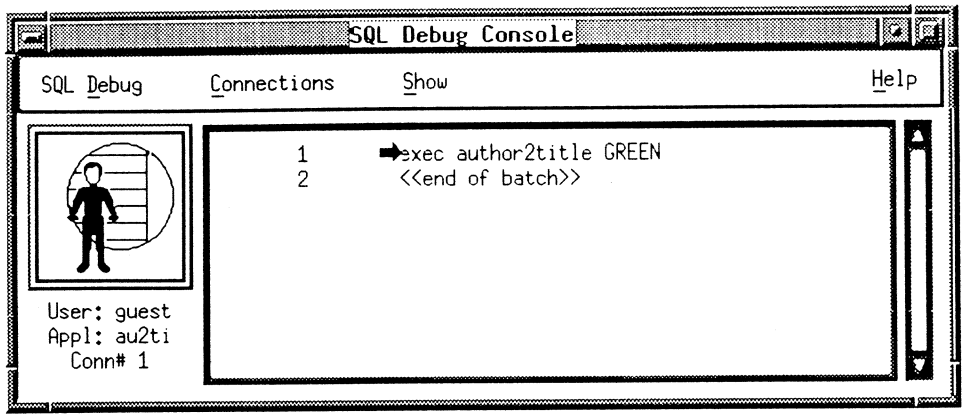
SYBASE

Systems Management Products

Sybase Confidential

Using SQL Debug

Transparent Debugging



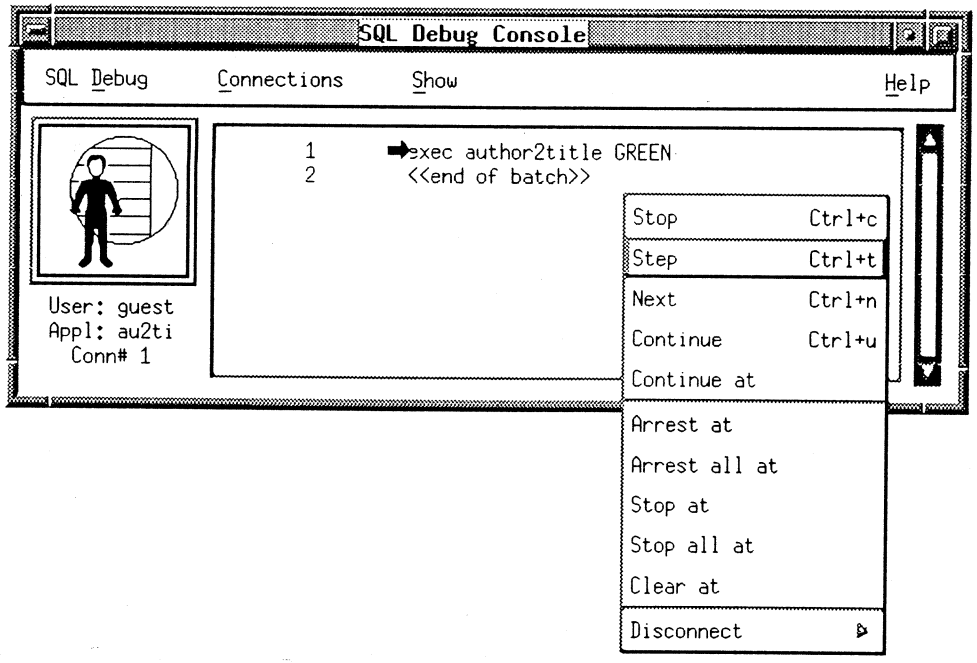
SYBASE

Systems Management Products

Sybase Confidential

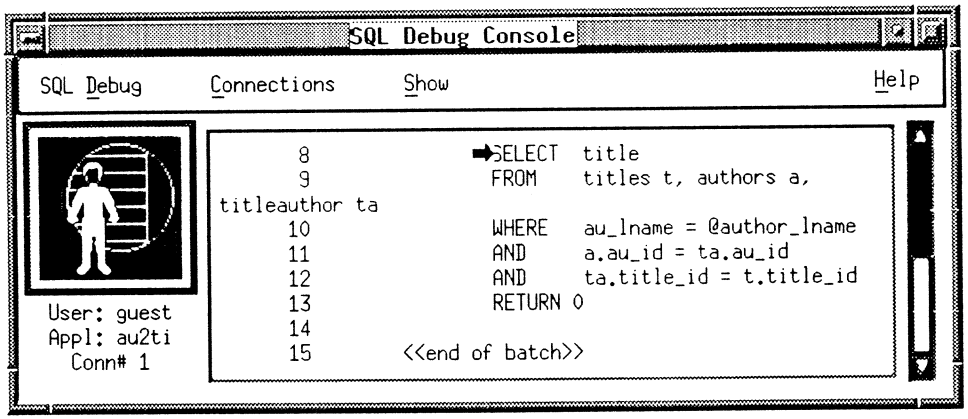
Using SQL Debug

Transparent Debugging



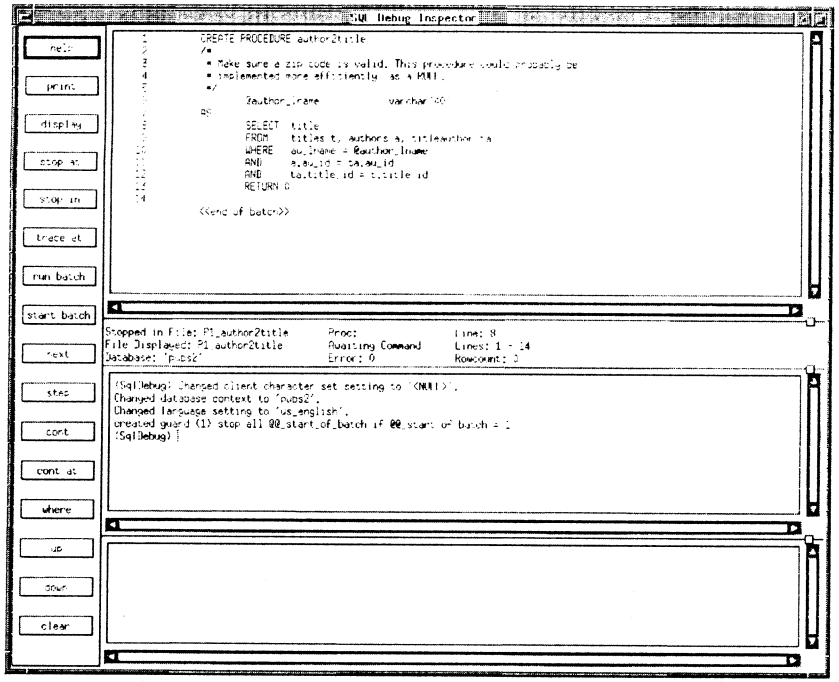
Using SQL Debug

Transparent Debugging



Using SQL Debug

Transparent Debugging



SYBASE

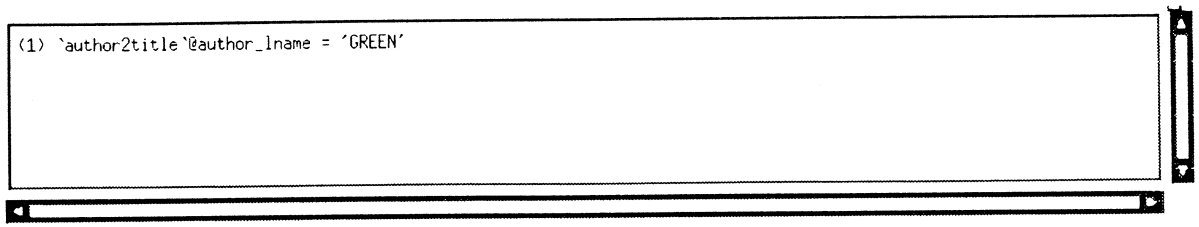
Systems Management Products

Sybase Confidential

Using SQL Debug

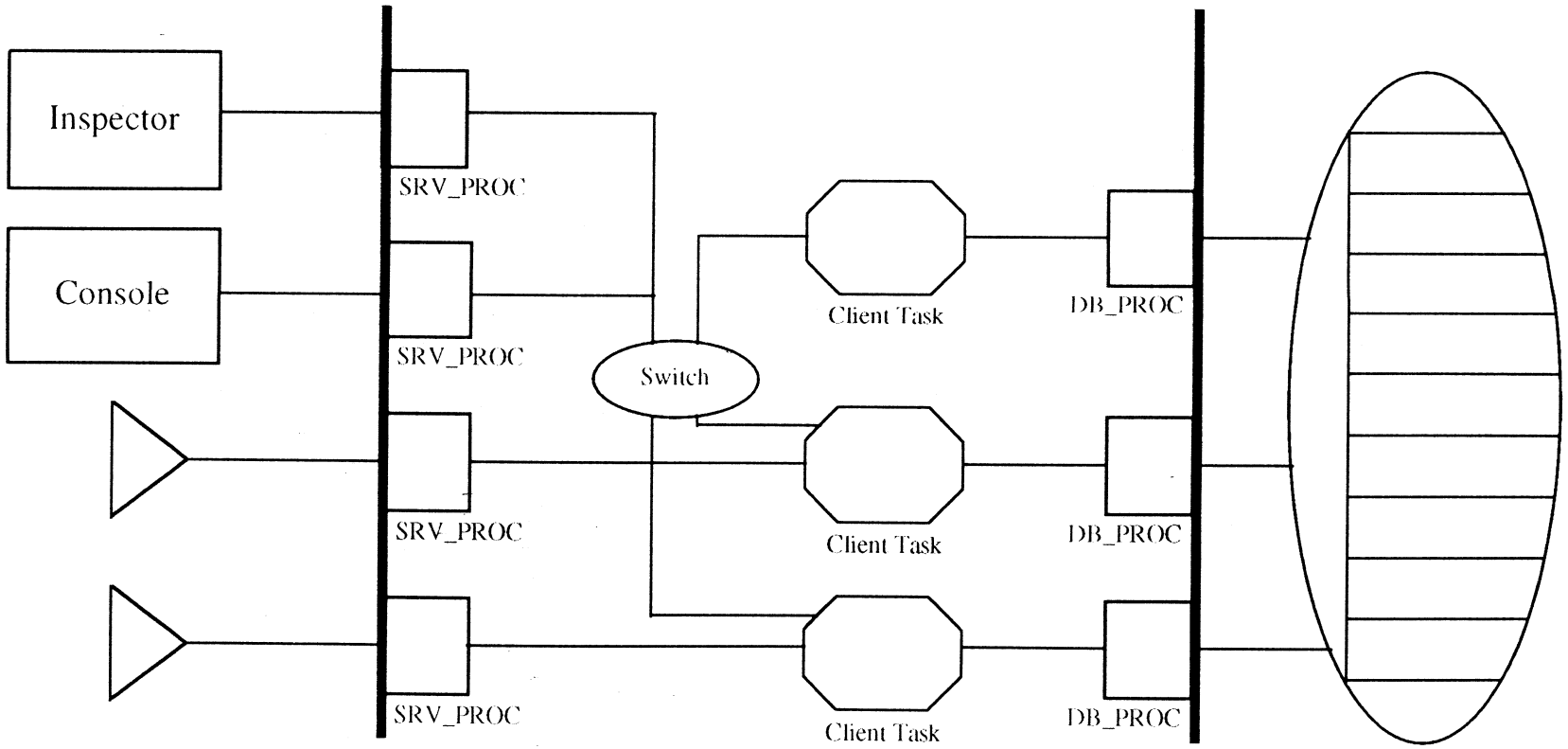
Transparent Debugging

```
(SqlDebug) display @author_lname
(SqlDebug)
```




```
(SqlDebug) step
(1 row affected)
title
----------------------------------------------------
(SqlDebug)
```

Internal Architecture

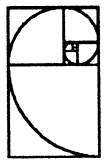


SYBASE

Systems Management Products

Availability

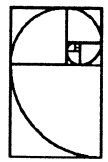
- Version 1.1.0
  Today
    - SQL Servers 4.0.1, 4.2, 4.8, 4.9 (Partial)
    - SunOS, AIX, HP-UX, VMS



SYBASE

Availability

- Version 10
  - 4Q93
    - SQL Servers 4.9, 10
    - SunOS, AIX, HP-UX
  - 1H93
    - Solaris, NCR/SVR4
  - 3Q94
    - AXP/VMS, AXP/OSF1



SYBASE

SYSTEM 10

Precompiler

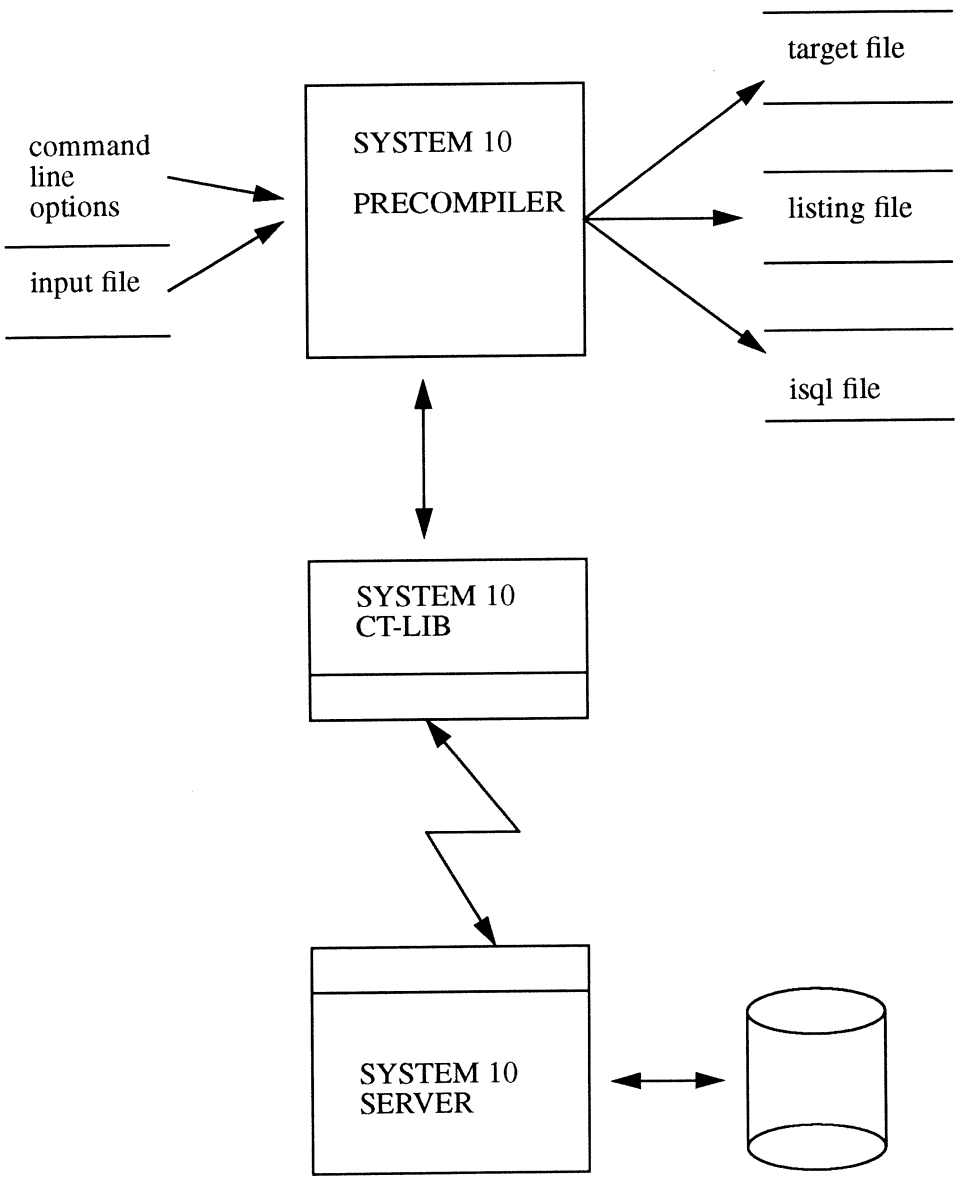
Sybase European Users' Conference

October 12-14, 1993

The Hague, Netherlands

By Jeff Richey

System Overview



Feature Set

Supported Languages

ANSI Compatibility (SQL-89, SQL-92)

Static SQL

Data Types

Host Language
- scalar
- array
- structure/union, group

SQL
- Text, Image, Numeric, Decimal, Money, Date, etc.

Dynamic SQL

Embedded Transact SQL

Diagnostics Handling

Stored Procedures

Internationalization

Ct-Lib Runtime Library

Misc:

- rebinding
- set fetch count on cursor
- array fetches
- include table/view statement

Modular Programming Feature

4.0.x, Oracle and DB2 compatibility and porting

Data Types - Host Language

Declaration Sections

multiple
scoping rules follows the native language scoping rules
host language declarations and initialization
SQL declarations
parameter declarations

Host Language data type to SQL data type conversion

C

char
int
float
double
[long, short, unsigned]
structures
unions
arrays [char, int, float, double]
#define
typedefs

Cobol

PIC X(s)	
PIC S9(n)	[COMP, COMP-4, COMP-5, BINARY]
PIC S9(n) V9(m)	[COMP-3 (packed decimal)]
	[DSLS, DSTS, DSL, DST (non-packed decimal)]
	1 <= n <= 9
	1 <= m <= 18

COMP-1, COMP-2 for VMS
DISPLAY SIGN LEADING SEPERATE
VALUES clause
REDEFINES
GLOBAL
EXTERNAL
groups
arrays

Data Types - SQL

CS_BINARY

CS_VARBINARY

CS_BIT

CS_CHAR

CS_VARCHAR

CS_DATETIME

CS_DATETIME4

CS_TINYINT

CS_SMALLINT

CS_INT

CS_DECIMAL

CS_NUMERIC

CS_FLOAT

CS_REAL

CS_MONEY

CS_MONEY4

CS_TEXT

CS_IMAGE

Dynamic SQL

Method 1

EXECUTE IMMEDIATE

Method 2

PREPARE
EXECUTE
DEALLOCATE PREPARE
Descriptor Usage (Allocate, Set, Get, Deallocate)

Method 3
PREPARE
DECLARE
OPEN
FETCH
UPDATE ... WHERE CURRENT OF
DELETE ... WHERE CURRENT OF
CLOSE
DEALLOCATE PREPARE

Method 4

PREPARE
DECLARE
ALLOCATE DESCRIPTOR
DESCRIBE INPUT
DESCRIBE OUTPUT
GET DESCRIPTOR
SET DESCRIPTOR
OPEN
FETCH
UPDATE ... WHERE CURRENT OF
DELETE ... WHERE CURRENT OF
CLOSE
DEALLOCATE DESCRIPTOR
DEALLOCATE PREPARE

Dynamic SQL Method 1: Source

```
void
dyn_m1()
{
        EXEC SQL BEGIN DECLARE SECTION;
            char    str1[255];
         EXEC SQL END DECLARE SECTION;

        printf("\n\nDynamic SQL Method 1\n");
         printf("enter in a non-Select SQL statement:");
        scanf("%[^/]", str1);

        EXEC SQL EXECUTE IMMEDIATE :str1;

         printf("Dynamic SQL Method 1 completed\n\n");

}
```

Dynamic SQL Method 2a: Source

```
void
dyn_m2a()
{
        EXEC SQL BEGIN DECLARE SECTION;
            int     cnt, occur, descnt;
            char    str2[255], char_buff[255];
        EXEC SQL END DECLARE SECTION;

       printf("\n\nDynamic SQL Method 2 (non-Select)\n");
       printf("Enter in a non-Select statement to modify the number of sales for");
       printf(" a certain type or types of books:");
        scanf("%[^/]", str2);

       printf("\nEnter in the number of ? in the SQL statement");
       scanf("%d", &occur);

       EXEC SQL PREPARE S1 FROM :str2;

       EXEC SQL ALLOCATE DESCRIPTOR DINOUT WITH MAX :occur;

       EXEC SQL GET DESCRIPTOR DINOUT :descnt = COUNT;

       for (cnt = 1; cnt <= descnt; cnt++)
       {
           printf("enter in the type of book:");
           scanf("%[^/]", char_buff);

           EXEC SQL SET DESCRIPTOR DINOUT VALUE :cnt TYPE = 1, LENGTH = 255,
                   DATA = :char_buff;
        }


       EXEC SQL EXECUTE S1 USING SQL DESCRIPTOR DINOUT;

        EXEC SQL DEALLOCATE PREPARE S1;

       EXEC SQL DEALLOCATE DESCRIPTOR DINOUT;

       printf("Dynamic SQL Method 2 (non-Select) completed\n");

}
```

Dynamic SQL Method 2b: Source

```
void dyn_m2b()
{
    EXEC SQL BEGIN DECLARE SECTION;
        int     cnt, occur, descnt;
        char    str2[255], char_buff[255];
    EXEC SQL END DECLARE SECTION;

    printf("\n\nDynamic SQL Method 2 (single-row Select)\n");
    printf("Enter in a Select statement that will return a single row");
    printf(" for a certain type of book:");
    scanf("%[^/]", str2);

    printf("\nEnter in the larger of the columns to be retrieved or the number ");
    printf("of ? in the SQL statement:\n");
    scanf("%d", &occur);

     EXEC SQL PREPARE S2 FROM :str2;

    EXEC SQL ALLOCATE DESCRIPTOR DINOUT WITH MAX :occur;

    EXEC SQL GET DESCRIPTOR DINOUT :descnt = COUNT;

    for (cnt = 1; cnt <= descnt; cnt++)
    {
        printf("enter in the type of book:");
        scanf("%[^/]", char_buff);

        EXEC SQL SET DESCRIPTOR DINOUT VALUE :cnt TYPE = 1, LENGTH = 255,
                DATA = :char_buff;
    }

     EXEC SQL EXECUTE S2 INTO SQL DESCRIPTOR DINOUT
                USING SQL DESCRIPTOR DINOUT;

     print_descriptor();

    EXEC SQL DEALLOCATE PREPARE S2;
    EXEC SQL DEALLOCATE DESCRIPTOR DINOUT;

    printf("Dynamic SQL Method 2 (single-row select) completed\n");

}
```

Dynamic SQL Method 3: Source

```
void dyn_m3()
{
    EXEC SQL BEGIN DECLARE SECTION;
        int     sales;
        char    str3[255], title[255];
    EXEC SQL END DECLARE SECTION;

    printf("\n\nDynamic SQL Method 3\n");
    printf("Enter in a Select statement to retrieve the list of books by");
    printf(" titles and the number of sales for a specified type(s):");
    scanf("%[^/]", str3);

    EXEC SQL PREPARE S3 FROM :str3;

    EXEC SQL DECLARE C1 CURSOR FOR S3;

    EXEC SQL OPEN C1;

    printf("title\t\t\t\t\t\t\t sales\n");
    printf("-------------------------------------------------------------");

    while (sqlca.sqlcode == 0)
    {
        EXEC SQL FETCH C1 INTO :title, :sales;

        printf("%s\t\t\t\t\t\t %d\n", title, sales);
    }

    EXEC SQL CLOSE C2;

    EXEC SQL DEALLOCATE PREPARE S7;

    printf("Dynamic SQL Method 3 completed\n\n");

}
```

Dynamic SQL Method 4: Source

```
void dyn_m4()
{
    EXEC SQL BEGIN DECLARE SECTION;
        int     occur;
        char    str4[255];
    EXEC SQL END DECLARE SECTION;

    printf("\n\nDynamic sql Method 4\n");
    printf("Enter in a Select statement to retrieve any kind of ");
    printf("information from the pubs database:");
    scanf("%[^/]", str4);

    printf("\nEnter in the larger of the columns to be retrieved or the number ");
    printf("of ? in the SQL statement:\n");
    scanf("%d", &occur);

     EXEC SQL PREPARE S4 FROM :str4;

    EXEC SQL DECLARE C2 CURSOR FOR S4;

    EXEC SQL DESCRIBE INPUT S4 USING SQL DESCRIPTOR DINOUT;

    fill_descriptor();

     EXEC SQL OPEN C2 USING SQL DESCRIPTOR DINOUT;

    EXEC SQL DESCRIBE OUTPUT S4 USING SQL DESCRIPTOR DINOUT;

     while (sqlca.sqlcode == 0)
     {
        EXEC SQL FETCH C2 INTO SQL DESCRIPTOR DINOUT;

        print_descriptor();
     }

     EXEC SQL CLOSE C2;

    EXEC SQL DEALLOCATE DESCRIPTOR DINOUT;
    EXEC SQL DEALLOCATE PREPARE S4;

    printf("Dynamic SQL Method 4 completed\n\n");
}
```

Dynamic SQL Descriptor Usage: Source

```
void
print_descriptor()
{
        EXEC SQL BEGIN DECLARE SECTION;
            int     int_buff, index_colcnt, coltype, descnt;
            char    char_buff[255], colname[255];
            ...
        EXEC SQL END DECLARE SECTION;

        EXEC SQL GET DESCRIPTOR DINOUT :descnt = COUNT;

       printf("Column name \t\tColumn data\n");
       printf("----------- \t\t----------------------------------------\n");

       for (index_colcnt = 1; index_colcnt <= descnt; index_colcnt++)
       { /* get each column attribute */
            EXEC SQL GET DESCRIPTOR DINOUT VALUE :index_colcnt :coltype = TYPE;

            switch(coltype)
            {
                case 1:     /* character type */
                    EXEC SQL GET DESCRIPTOR DINOUT VALUE :index_colcnt
                            :colname = NAME, :char_buff = DATA;
                    printf("%s \t\t %s\n", colname, char_buff);
                    break;

                case 4:     /* integer type */
                    EXEC SQL GET DESCRIPTOR DINOUT VALUE :index_colcnt
                            :colname = NAME, :int_buff = DATA;
                    printf("%s \t\t %d\n", colname, int_buff);
                    break;

                ....
        }
  }

 }
```

Dynamic SQL Descriptor Usage: Source (cont.)

```
void
fill_descriptor()
{
    EXEC SQL BEGIN DECLARE SECTION;
        int         cnt, coltype, descnt;
        char        colname[255];
        short       smallint_buff;
        int         int_buff;
        float       real_buff;
        double      doubleprec_buff, float_buff;
        char        char_buff[255];
        ...
        CS_NUMERIC numeric_buff;
        CS_DECIMAL decimal_buff;
    EXEC SQL END DECLARE SECTION;

    EXEC SQL GET DESCRIPTOR DINOUT :descnt = COUNT;

    for (cnt = 1; cnt <= descnt; cnt++)
    {
        printf("Enter in the data type of the %d ?:", cnt);
        scanf("%d", &coltype);

        switch(coltype)
        {
            case 1:/* character type */
                printf("Enter in the value of the data:");
                scanf("%[^/]", char_buff);
                EXEC SQL SET DESCRIPTOR DINOUT VALUE :cnt TYPE = 1,
                        DATA = :char_buff;
                break;

            case 2:/* numeric type */
                printf("Enter in the value of the data:");
                scanf("%d", &numeric_buff);
                EXEC SQL SET DESCRIPTOR DINOUT VALUE :cnt TYPE = :coltype,
                        DATA = :numeric_buff;
                break;
            ...
        }
    }
}
```

Embedded Transact SQL

Method 1 (Non-Stored Procedure)

EXECUTE <status parameter> PY(<argument list>)

CREATE PROCEDURE PZ(<parameter list>)
AS ....

CREATE DEFAULT
CREATE RULE
CREATE TRIGGER
CREATE VIEW
USE STATEMENT

Method 2 (Stored Procedure Cursor Result Set)

DECLARE CURSOR ... FOR EXECUTE P1(<argument list>)
OPEN
FETCH
UPDATE ... WHERE CURRENT OF
DELETE ... WHERE CURRENT OF
CLOSE

Method 3 (Batch or Multiple Result Sets)

similiar to Dynamic SQL Method 4

Embedded Transact SQL

```
void
tsql_1()
{
    EXEC SQL BEGIN DECLARE SECTION;
i       char    ename[30], addr[30];
        int     ennum, dept, mgr;
    EXEC SQL END DECLARE SECTION;

    EXEC SQL USE my_database;

    EXEC SQL EXECUTE SQLCODE = tsql_smpl1;

    EXEC SQL EXECUTE SQLCODE = tsql_smpl2(@v1 = :ename OUTPUT,
                    @v2 = :ennum OUT, @v3 = :addr OUTPUT,
                    @v4 = :dept OUT, @v5 = :mgr OUTPUT);

    printf("employee name \t number \t department \t manager \t address \n");
    printf("--------------- \t ------ \t ---------- \t ------- \t -------\n");
    printf("%s \t %d \t %d \t %d \t %s\n\n\n", ename, ennum, dept, mgr, addr);

    EXEC SQL CREATE TRIGGER my_trigger
            ON employee FOR UPDATE AS
            IF UPDATE(salary)
            BEGIN
                UPDATE employee
                    SET salary = salary + salary * .1
                    WHERE title = 'engineer';

    EXEC SQL CREATE PROCEDURE tsql_smpl9(@deptno INT, @empno INT)
        AS
            declare @empsal decimal(7,2)
            select @empsal = max(salary) from employee
            update employee set salary = @empsal
                where empnum = @empno and deptnum = @deptno;
```

Embedded Transact SQL Single Result Set: Source

```
void
tsql_3()
{
      printf("executing a stored procedure that returns a single result set\n");
      EXEC SQL DECLARE C1 CURSOR FOR EXECUTE tsql_smpl3();

      EXEC SQL OPEN C1;

      printf("employee name \t number \t department \t manager \t address \n");
      printf("--------------- \t ------ \t ---------- \t ------- \t -------\n");

      while (sqlca.sqlcode == 0)
      {
          EXEC SQL FETCH C1 INTO :ename, :ennum, :addr, :dept, :mgr;
      printf("%s \t %d \t %d \t %d \t %s\n", ename, ennum, dept, mgr, addr);
      }

      printf("\n\n");

      EXEC SQL CLOSE C1;
}
```

Embedded Transact SQL: Multiple Result Set

Not Supported Thru Cursors

Support Thru parameters

tsql_*.pc

```
--------
...
EXEC SQL EXECUTE SQLCODE = tsql_mrs(@numid_1 = :empnum_1,
                    @numid_2 = :empnum_2, @enum_1 = :empnum_1 out,
                    @enum_2 = :empnum_2 out, @ename_1 = :ename_1 out,
                    @ename_2 = :ename_2 out, @dname_1 = :dname_1 out,
                    @dname_2 = :dname_2 out, @manager_1 = :manager_1 out,
                    @manager_2 = :manager_2 out);

...
```

tsql_*.sql

```
---------
...
CREATE PROCEDURE tsql_mrs(@numid_1 int, @numid_2 int, @enum_1 int out,
                          @enum_2 int out, @ename_1 char(30) out,
                          @ename_2 char(30) out, @dname_1 char(30) out,
                          @dname_2 char(30) out, @manager_1 int out,
                          @manager_2 int out)
    AS
    SELECT @enum_1=e.empnum, @ename_1=e.empname,
           @dname_1=d.deptname, @manager_1=d.manager
            FROM employee e, department d
            WHERE e.deptnum = d.deptnum AND e.empnum = @numid_1

    SELECT @enum_2=e.empnum, @ename_2=e.empname,
           @dname_2=d.deptname, @manager_2=d.manager
            FROM employee e, department d
            WHERE e.deptnum = d.deptnum AND e.empnum = @numid_2;

...
```

Diagnostics Handling

SQL Whenever Statement

SQLCA

SQLCODE

SQLSTATE

SQL Get Diagnostics Statement (multiple messages)

Diagnostics Handling: Source

```
EXEC SQL WHENEVER SQLERROR CALL error_handler();

...

 diag_cnt = 20;
 EXEC SQL SET TRANSACTION DIAGNOSTICS SIZE :diag_cnt;


...


void
error_handler()
{
        EXEC SQL GET DIAGNOSTICS :num_msgs = NUMBER;
       for (condcnt=0; condcnt < num_msgs; condcnt++)
       {
              EXEC SQL GET DIAGNOSTICS EXCEPTION :condcnt
                            :sqlca = SQLCA_INFO;
              printf("error text:%s\n,error code: %d\n",
                            sqlca.sqlerrm.sqlerrmc, sqlca.sqlcode);

              ...
       }

       ...

}



       if (sqlca.sqlcode != 0)
              error_handler();


       if (SQLCODE != 0)
              error_handler();


       if (strncmp(SQLSTATE, "00", 2) != 0)
              error_handler();
```

Misc. Features

Setting the Fetch Count on a Cursor

```
EXEC SQL OPEN C1 [ ROW_COUNT = 20 ] [ USING ...]
```

Rebinding

```
EXEC SQL FETCH [ NOREBIND ] C1 INTO ....
```

Array Fetches

```
EXEC SQL BEGIN DECLARE SECTION;
    ...
    int             empno[90];
    ...
EXEC SQL END DECLARE SECTION;

...

EXEC SQL SELECT empnumber INTO :empno FROM employee
            WHERE ...
```

Include Table/View

```
EXEC SQL INCLUDE TABLE employee AS my_employee FIELD "my";


EXEC SQL FETCH INTO :my_employee;


typedef struct {
    int             my_empnum;
    char            my_name[30];
    char            my_address[30];
    int             my_manager;
} my_employee;
```

Modular Programming Feature

What is it?

Why would someone want to use it?

Examples:

```
void
mpf_prepare_statement(stmt_name, stmt_text)
    EXEC SQL BEGIN DECLARE SECTION;
        char * stmt_name;
        char * stmt_text;
    EXEC SQL END DECLARE SECTION;
{
        EXEC SQL PREPARE :stmt_name FROM :stmt_text;
}

void
mpf_open_cursor(cursor_name, row_cnt, descriptor_name)
    EXEC SQL BEGIN DECLARE SECTION;
        char * cursor_name;
        int rowcnt;
        char * descriptor_name;
    EXEC SQL END DECLARE SECTION;
{
        EXEC SQL OPEN :cursor_name ROW_COUNT = :rowcnt
                                        USING SQL DESCRIPTOR :descriptor_name;
}

void
mpf_fetch_cursor(cursor_name, descriptor_name)
    EXEC SQL BEGIN DECLARE SECTION;
        char * cursor_name;
        char * descriptor_name;
    EXEC SQL END DECLARE SECTION;
{
        EXEC SQL FETCH :cursor_name INTO SQL DESCRIPTOR :descriptor_name;
}
```

Porting Applications (Relational and Non-Relation) to Sybase

Application Migration

1. Modify (or create) file with ESQL statements
2. precompile, compile, link
3. execute against test data

Data Migration

1. determine strategy
2. determine which Sybase product to use
3. define database definitions (table, databases, indexes, etc.)
4. migrate the data
5. system test

Transaction Management/Locking

1. differences in transaction model
2. lock modes
3. simulate application

WHICH ONE

Embedded SQL

Call Level Interface (CT-Lib)

Module Language SQL

Database Programming Language (for SQL)

Appendices

Data Flow

Input File

Target File

ISQL File

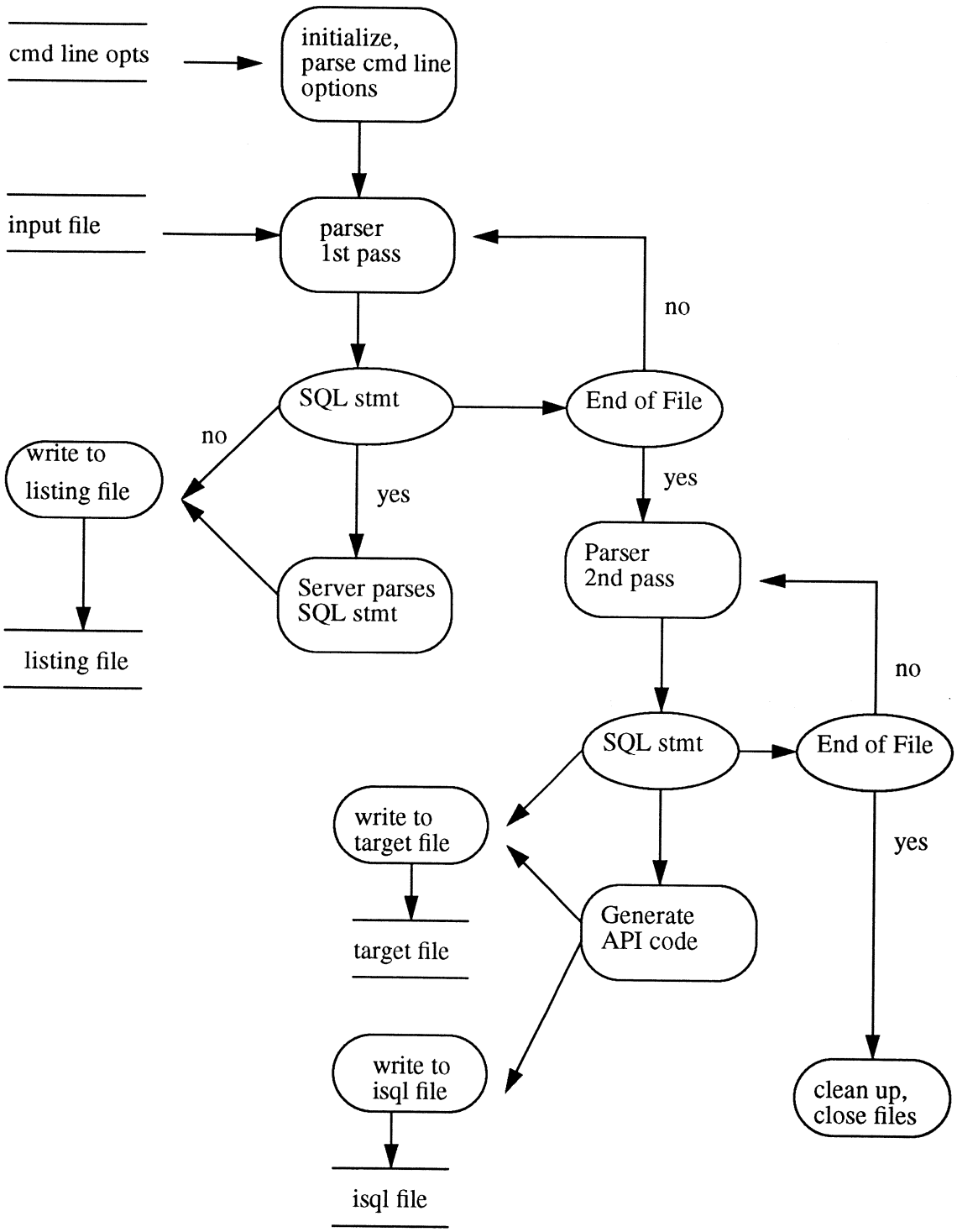
Listing File

Dynamic SQL example

Stored Procedure example

Modular Programming Feature example

SYSTEM 10 PRECOMPILER Data Flow



Input File

```
extern void main();
extern void init_table();
extern void cursor_sql();
extern void cleanup();
extern void error_handler();

EXEC SQL WHENEVER SQLERROR CALL error_handler();

EXEC SQL INCLUDE SQLCA;

void
main()
{
 EXEC SQL BEGIN DECLARE SECTION;
     char   user_id[30], pass_id[30], server_name[30];
     int    diag_cnt;
 EXEC SQL END DECLARE SECTION;

 strcpy(user_id, "sa");
 strcpy(pass_id, "");
 printf("enter in your server name: ");
 scanf("%s[^\n]", server_name);
 EXEC SQL CONNECT :user_id IDENTIFIED BY :pass_id USING :server_name;

 diag_cnt = 20;
 EXEC SQL SET TRANSACTION DIAGNOSTICS SIZE :diag_cnt;

 init_table();
 cursor_sql();
 cleanup();

 EXEC SQL DISCONNECT ALL;

}


void
error_handler()
{
 EXEC SQL BEGIN DECLARE SECTION;
     int    num_msgs, condcnt;
 EXEC SQL END DECLARE SECTION;
```

```
 EXEC SQL GET DIAGNOSTICS :num_msgs = NUMBER;
 for (condcnt=0; condcnt < num_msgs; condcnt++)
 {
     EXEC SQL GET DIAGNOSTICS EXCEPTION :condcnt
             :sqlca = SQLCA_INFO;
     printf("sqlcode is: %d\n message text: %s\n", sqlca.sqlcode,
         sqlca.sqlerrm.sqlerrmc);
 }

}


void
init_table()
{

 EXEC SQL CREATE TABLE employee(empnum integer, name char(30), address
                 char(30), manager integer, department integer);

 EXEC SQL INSERT INTO employee
         VALUES(1, "bill", "20 anything", 1001, 99);


 EXEC SQL INSERT INTO employee
         VALUES(2, "al", "21 anything", 1001, 99);

 EXEC SQL INSERT INTO employee
         VALUES(3, "george", "22 anything", 1001, 99);

}


void
cleanup()
{

 EXEC SQL ROLLBACK WORK;

 EXEC SQL DROP TABLE employee;

}


void
cursor_sql()
{
```

```
EXEC SQL BEGIN DECLARE SECTION;
    char   ename[30], addr[30];
    int    ennum, mgr, dept;
EXEC SQL END DECLARE SECTION;

printf("beginning of sql cursor operations exercise\n");

EXEC SQL DECLARE C1 CURSOR FOR Select * FROM employee WHERE name >=
:ename;

strcpy(ename, "al");
EXEC SQL OPEN C1 USING :ename;

printf("employee name   \t number \t department \t manager \t address \n");
printf("--------------- \t ------ \t ---------- \t ------- \t ----------\n");
while (sqlca.sqlcode >= 0)
{
    EXEC SQL FETCH C1 INTO :ennum, :ename, :addr, :mgr, :dept;
    printf("%s\t %d \t %d \t %d \t %s\n\n", ename, ennum, dept, mgr, addr);

}

EXEC SQL CLOSE C1;

printf("end of sql cursor operations exercise\n");

}
```

Target File

```
/*
** Generated code begins here.
*/
#include <sybhesql.h>
#include <sybtesql.h>
static CS_DATAFMT _sqldfmtCS_CHAR_TYPE = {
   "", 0, CS_CHAR_TYPE, (CS_FMT_NULLTERM | CS_FMT_PADBLANK), 0, CS_U-
NUSED,
      CS_UNUSED, 0, 1, 0, 0
}
;
static CS_DATAFMT _sqldfmtCS_INT_TYPE = {
   "", 0, CS_INT_TYPE, CS_FMT_UNUSED, 0, CS_UNUSED, CS_UNUSED, 0, 1, 0, 0
}
;

/*
** Generated code ends here.
*/
extern void main();
extern void init_table();
extern void cursor_sql();
extern void cleanup();
extern void error_handler();


/*
** SQL STATEMENT: 1
** EXEC SQL WHENEVER SQLERROR CALL error_handler();
*/


/*
** SQL STATEMENT: 2
** EXEC SQL INCLUDE SQLCA;
*/
static SQLCA sqlca;

/*
** Generated code ends here.
*/
```

```
void
main()
{

 /*
 ** SQL STATEMENT: 2
 ** EXEC SQL BEGIN DECLARE SECTION;
 */

     char    user_id[30], pass_id[30], server_name[30];
     int     diag_cnt;

 /*
 ** SQL STATEMENT: 3
 ** EXEC SQL END DECLARE SECTION;
 */

 /*
 ** Generated code ends here.
 */


 strcpy(user_id, "sa");
 strcpy(pass_id, "");
 printf("enter in your server name: ");
 scanf("%s[^\n]", server_name);

 /*
 ** SQL STATEMENT: 4
 ** EXEC SQL CONNECT :user_id IDENTIFIED BY :pass_id USING :server_name;
 */
 _sqlhandles.connName.lnlen = CS_NULLTERM;
 strcpy(_sqlhandles.connName.last_name, server_name);
 if ((_sqlretcode = _sqlprolog(&_sqlhandles, _SQL_CONNECT_TYPE, _SQL_CREATE,
    &sqlca, (CS_INT *)NULL, (CS_CHAR *)NULL)) == CS_SUCCEED)
 {
    if (_sqlhandles.doDecl == CS_TRUE)
    {
       _sqlretcode = ct_init(_sqlhandles.ctx, CS_VERSION_100);
       if (_sqlretcode == CS_SUCCEED)
       {
          _sqlretcode = ct_con_alloc(_sqlhandles.ctx,
             &_sqlhandles.conn.connection);
       }
```

```
if (_sqlretcode == CS_SUCCEED)
{
   _sqlretcode = ct_diag(_sqlhandles.conn.connection, CS_INIT,
      CS_UNUSED, CS_UNUSED, NULL);
   if (_sqlretcode != CS_SUCCEED)
   {
      _sqlsetintrerr(&sqlca, (CS_INT *)NULL, (CS_CHAR *)NULL,
         (CS_INT)_SQL_INTRERR_25002);
   }

}

if (_sqlretcode == CS_SUCCEED)
{
   _sqlretcode = ct_cmd_alloc(_sqlhandles.conn.connection,
      &_sqlhandles.conn.command);
}

if (_sqlretcode == CS_SUCCEED)
{
   _sqlretcode = _sqlctdiag(_sqlhandles.conn.connection, CS_CLEAR,
      CS_UNUSED, &sqlca, (CS_INT *)NULL, (CS_CHAR *)NULL);
}

if (_sqlretcode == CS_SUCCEED)
{
   _sqlretcode = ct_capability(_sqlhandles.conn.connection, CS_SET,
      CS_CAP_RESPONSE, CS_RES_NOSTRIPBLANKS, &_sqltrue);
}

if (_sqlretcode == CS_SUCCEED)
{
   _sqlretcode = ct_con_props(_sqlhandles.conn.connection, CS_SET,
      CS_USERNAME, user_id, CS_NULLTERM, NULL);
}

if (_sqlretcode == CS_SUCCEED)
{
   _sqlretcode = ct_con_props(_sqlhandles.conn.connection, CS_SET,
      CS_EXTRA_INF, &_sqltrue, CS_UNUSED, NULL);
}

if (_sqlretcode == CS_SUCCEED)
{
   _sqlretcode = ct_con_props(_sqlhandles.conn.connection, CS_SET,
      CS_ANSI_BINDS, &_sqltrue, CS_UNUSED, NULL);
```

```
}

if (_sqlretcode == CS_SUCCEED)
{
   _sqlretcode = ct_con_props(_sqlhandles.conn.connection, CS_SET,
      CS_PASSWORD, pass_id, CS_NULLTERM, NULL);
}

if (_sqlretcode == CS_SUCCEED)
{
   _sqlretcode = ct_connect(_sqlhandles.conn.connection,
      server_name, CS_NULLTERM);
}

if (_sqlretcode == CS_SUCCEED)
{
   _sqlretcode = ct_options(_sqlhandles.conn.connection, CS_SET,
      CS_OPT_ANSINULL, &_sqltrue, CS_UNUSED, NULL);
}

if (_sqlretcode == CS_SUCCEED)
{
   _sqlretcode = ct_options(_sqlhandles.conn.connection, CS_SET,
      CS_OPT_ISOLATION, &_sqllevel3, CS_UNUSED, NULL);
}

if (_sqlretcode == CS_SUCCEED)
{
   _sqlretcode = ct_options(_sqlhandles.conn.connection, CS_SET,
      CS_OPT_CHAINXACTS, &_sqltrue, CS_UNUSED, NULL);
}

if (_sqlretcode == CS_SUCCEED)
{
   _sqlhandles.connName.thinkexists = CS_FALSE;
   _sqlretcode = cs_objects(_sqlhandles.ctx, CS_SET,
      &_sqlhandles.connName, &_sqlhandles.conn);
   if (_sqlretcode == CS_SUCCEED)
   {
      _sqlhandles.current.buffer = _sqlhandles.connName.last_name;
      _sqlhandles.current.buflen = _sqlhandles.connName.lnlen;
      _sqlretcode = cs_objects(_sqlhandles.ctx, CS_CLEAR,
         &_sqlhandles.currentName, &_sqlhandles.current);
      _sqlhandles.current.buffer = _sqlhandles.connName.last_name;
      _sqlhandles.current.buflen = _sqlhandles.connName.lnlen;
      _sqlretcode = cs_objects(_sqlhandles.ctx, CS_SET,
```

```
            &_sqlhandles.currentName, &_sqlhandles.current);
        }

      }

      _sqlretcode = _sqlepilog(&_sqlhandles, _SQL_CONNECT_TYPE, &sqlca,
        (CS_INT *)NULL, (CS_CHAR *)NULL);
    }

  }

  if (sqlca.sqlcode < 0)
  {
    error_handler();
  }


  /*
  ** Generated code ends here.
  */


  diag_cnt = 20;

  /*
  ** SQL STATEMENT: 5
  ** EXEC SQL SET TRANSACTION DIAGNOSTICS SIZE :diag_cnt;
  */
  if ((_sqlretcode = ct_diag(_sqlhandles.conn.connection, CS_MSGLIMIT,
    CS_ALLMSG_TYPE, CS_UNUSED, &diag_cnt)) == CS_FAIL)
  {
    _sqlretcode = _SQL_INTRERR_25002;
  }


  /*
  ** Generated code ends here.
  */


  init_table();
  cursor_sql();
  cleanup();


  /*
```

```
** SQL STATEMENT: 6
** EXEC SQL DISCONNECT ALL;
*/
_sqlretcode = cs_ctx_global(CS_VERSION_100, &_sqlhandles.ctx);
if (_sqlretcode != CS_SUCCEED)
{
   _sqlsetintrerr(&sqlca, (CS_INT *)NULL, (CS_CHAR *)NULL,
      (CS_INT)_SQL_INTRERR_25001);
}

if (_sqlretcode == CS_SUCCEED)
{
   _sqlretcode = ct_exit(_sqlhandles.ctx, CS_UNUSED);
   if (_sqlretcode != CS_SUCCEED)
   {
      _sqlretcode = ct_exit(_sqlhandles.ctx, CS_FORCE_EXIT);
   }

   _sqlretcode = cs_ctx_drop(_sqlhandles.ctx);
   _sqlretcode = cs_ctx_global(CS_VERSION_100, &_sqlhandles.ctx);
}

if (sqlca.sqlcode < 0)
{
   error_handler();
}


/*
** Generated code ends here.
*/


}


void
error_handler()
{

 /*
 ** SQL STATEMENT: 6
 ** EXEC SQL BEGIN DECLARE SECTION;
 */

     int    num_msgs, condcnt;
```

```
/*
** SQL STATEMENT: 7
** EXEC SQL END DECLARE SECTION;
*/

/*
** Generated code ends here.
*/



/*
** SQL STATEMENT: 8
** EXEC SQL GET DIAGNOSTICS :num_msgs = NUMBER;
*/
if ((_sqlretcode = ct_diag(_sqlhandles.conn.connection, CS_STATUS,
   CS_ALLMSG_TYPE, CS_UNUSED, &num_msgs)) == CS_FAIL)
{
   _sqlretcode = _SQL_INTRERR_25002;
}


/*
** Generated code ends here.
*/

for (condcnt=0; condcnt < num_msgs; condcnt++)
{

      /*
      ** SQL STATEMENT: 9
      ** EXEC SQL GET DIAGNOSTICS EXCEPTION :condcnt
      **                :sqlca = SQLCA_INFO;
      */
      if ((_sqlretcode = ct_diag(_sqlhandles.conn.connection, CS_GET,
         SQLCA_TYPE, condcnt, &sqlca)) == CS_FAIL)
      {
         _sqlretcode = _SQL_INTRERR_25002;
      }


      /*
      ** Generated code ends here.
      */
```

```
    printf("sqlcode is: %d\n message text: %s\n", sqlca.sqlcode,
        sqlca.sqlerrm.sqlerrmc);
 }

}


void
init_table()
{


 /*
 ** SQL STATEMENT: 10
 ** EXEC SQL CREATE TABLE employee(empnum integer, name char(30), address
 **                 char(30), manager integer, department integer);
 */
 _sqlhandles.connName.lnlen = CS_UNUSED;
 if ((_sqlretcode = _sqlprolog(&_sqlhandles, _SQL_CONNECT_TYPE, _SQL_GET,
    &sqlca, (CS_INT *)NULL, (CS_CHAR *)NULL)) == CS_SUCCEED)
 {
    _sqlretcode = ct_command(_sqlhandles.conn.command, CS_LANG_CMD, "CREATE T"
"ABLE employee(empnum integer, name char(30), address "
    "                 char(30), manager integer, department integer"
")", 112, CS_UNUSED);
    _sqlretcode = ct_send(_sqlhandles.conn.command);
    while ((_sqlretcode = ct_results(_sqlhandles.conn.command,
       &_sqlrestype)) == CS_SUCCEED)
    {
       ;
    }

    if (_sqlretcode != CS_END_RESULTS)
    {
       _sqlretcode = ct_cancel(NULL, _sqlhandles.conn.command,
          CS_CANCEL_ALL);
    }
     else {
       _sqlretcode = CS_SUCCEED;
    }

    _sqlretcode = _sqlepilog(&_sqlhandles, _SQL_CONNECT_TYPE, &sqlca,
       (CS_INT *)NULL, (CS_CHAR *)NULL);
 }

 if (sqlca.sqlcode < 0)
```

```
{
   error_handler();
}


/*
** Generated code ends here.
*/



/*
** SQL STATEMENT: 11
** EXEC SQL INSERT INTO employee
**          VALUES(1, "bill", "20 anything", 1001, 99);
*/
_sqlhandles.connName.lnlen = CS_UNUSED;
if ((_sqlretcode = _sqlprolog(&_sqlhandles, _SQL_CONNECT_TYPE, _SQL_GET,
   &sqlca, (CS_INT *)NULL, (CS_CHAR *)NULL)) == CS_SUCCEED)
{
   _sqlretcode = ct_command(_sqlhandles.conn.command, CS_LANG_CMD, "INSERT I"
"NTO employee "
   "          VALUES(1, \"bill\", \"20 anything\", 1001, 99)", 67,
      CS_UNUSED);
   _sqlretcode = ct_send(_sqlhandles.conn.command);
   while ((_sqlretcode = ct_results(_sqlhandles.conn.command,
      &_sqlrestype)) == CS_SUCCEED)
   {
      ;
   }

   if (_sqlretcode != CS_END_RESULTS)
   {
      _sqlretcode = ct_cancel(NULL, _sqlhandles.conn.command,
         CS_CANCEL_ALL);
   }
    else {
      _sqlretcode = CS_SUCCEED;
   }

   _sqlretcode = _sqlepilog(&_sqlhandles, _SQL_CONNECT_TYPE, &sqlca,
      (CS_INT *)NULL, (CS_CHAR *)NULL);
}

if (sqlca.sqlcode < 0)
{
```

```
    error_handler();
}


/*
** Generated code ends here.
*/




/*
** SQL STATEMENT: 12
** EXEC SQL INSERT INTO employee
**          VALUES(2, "al", "21 anything", 1001, 99);
*/
_sqlhandles.connName.lnlen = CS_UNUSED;
if ((_sqlretcode = _sqlprolog(&_sqlhandles, _SQL_CONNECT_TYPE, _SQL_GET,
    &sqlca, (CS_INT *)NULL, (CS_CHAR *)NULL)) == CS_SUCCEED)
{
    _sqlretcode = ct_command(_sqlhandles.conn.command, CS_LANG_CMD, "INSERT I"
"NTO employee "
    "               VALUES(2, \"al\", \"21 anything\", 1001, 99)", 65,
        CS_UNUSED);
    _sqlretcode = ct_send(_sqlhandles.conn.command);
    while ((_sqlretcode = ct_results(_sqlhandles.conn.command,
        &_sqlrestype)) == CS_SUCCEED)
    {
        ;
    }

    if (_sqlretcode != CS_END_RESULTS)
    {
        _sqlretcode = ct_cancel(NULL, _sqlhandles.conn.command,
            CS_CANCEL_ALL);
    }
    else {
        _sqlretcode = CS_SUCCEED;
    }

    _sqlretcode = _sqlepilog(&_sqlhandles, _SQL_CONNECT_TYPE, &sqlca,
        (CS_INT *)NULL, (CS_CHAR *)NULL);
}

if (sqlca.sqlcode < 0)
{
```

```
    error_handler();
  }


  /*
  ** Generated code ends here.
  */



  /*
  ** SQL STATEMENT: 13
  ** EXEC SQL INSERT INTO employee
  **          VALUES(3, "george", "22 anything", 1001, 99);
  */
  _sqlhandles.connName.lnlen = CS_UNUSED;
  if ((_sqlretcode = _sqlprolog(&_sqlhandles, _SQL_CONNECT_TYPE, _SQL_GET,
     &sqlca, (CS_INT *)NULL, (CS_CHAR *)NULL)) == CS_SUCCEED)
  {
     _sqlretcode = ct_command(_sqlhandles.conn.command, CS_LANG_CMD, "INSERT I"
"NTO employee"
     "          VALUES(3, \"george\", \"22 anything\", 1001, 99)", 68,
        CS_UNUSED);
     _sqlretcode = ct_send(_sqlhandles.conn.command);
     while ((_sqlretcode = ct_results(_sqlhandles.conn.command,
        &_sqlrestype)) == CS_SUCCEED)
     {
        ;
     }

     if (_sqlretcode != CS_END_RESULTS)
     {
        _sqlretcode = ct_cancel(NULL, _sqlhandles.conn.command,
           CS_CANCEL_ALL);
     }
     else {
        _sqlretcode = CS_SUCCEED;
     }

     _sqlretcode = _sqlepilog(&_sqlhandles, _SQL_CONNECT_TYPE, &sqlca,
        (CS_INT *)NULL, (CS_CHAR *)NULL);
  }

  if (sqlca.sqlcode < 0)
  {
     error_handler();
```

```
 }


 /*
 ** Generated code ends here.
 */


}


void
cleanup()
{


 /*
 ** SQL STATEMENT: 14
 ** EXEC SQL ROLLBACK WORK;
 */
 _sqlhandles.connName.lnlen = CS_UNUSED;
 if ((_sqlretcode = _sqlprolog(&_sqlhandles, _SQL_CONNECT_TYPE, _SQL_GET,
    &sqlca, (CS_INT *)NULL, (CS_CHAR *)NULL)) == CS_SUCCEED)
 {
    _sqlretcode = ct_command(_sqlhandles.conn.command, CS_LANG_CMD, "ROLLBACK"
" WORK", 13, CS_UNUSED);
    _sqlretcode = ct_send(_sqlhandles.conn.command);
    while ((_sqlretcode = ct_results(_sqlhandles.conn.command,
       &_sqlrestype)) == CS_SUCCEED)
    {
       ;
    }

    if (_sqlretcode != CS_END_RESULTS)
    {
       _sqlretcode = ct_cancel(NULL, _sqlhandles.conn.command,
          CS_CANCEL_ALL);
    }
     else {
       _sqlretcode = CS_SUCCEED;
    }

    _sqlretcode = _sqlepilog(&_sqlhandles, _SQL_CONNECT_TYPE, &sqlca,
       (CS_INT *)NULL, (CS_CHAR *)NULL);
 }
```

```
if (sqlca.sqlcode < 0)
{
   error_handler();
}


/*
** Generated code ends here.
*/



/*
** SQL STATEMENT: 15
** EXEC SQL DROP TABLE employee;
*/
_sqlhandles.connName.lnlen = CS_UNUSED;
if ((_sqlretcode = _sqlprolog(&_sqlhandles, _SQL_CONNECT_TYPE, _SQL_GET,
   &sqlca, (CS_INT *)NULL, (CS_CHAR *)NULL)) == CS_SUCCEED)
{
   _sqlretcode = ct_command(_sqlhandles.conn.command, CS_LANG_CMD, "DROP TAB"
"LE employee", 19, CS_UNUSED);
   _sqlretcode = ct_send(_sqlhandles.conn.command);
   while ((_sqlretcode = ct_results(_sqlhandles.conn.command,
      &_sqlrestype)) == CS_SUCCEED)
   {
      ;
   }

   if (_sqlretcode != CS_END_RESULTS)
   {
      _sqlretcode = ct_cancel(NULL, _sqlhandles.conn.command,
         CS_CANCEL_ALL);
   }
   else {
      _sqlretcode = CS_SUCCEED;
   }

   _sqlretcode = _sqlepilog(&_sqlhandles, _SQL_CONNECT_TYPE, &sqlca,
      (CS_INT *)NULL, (CS_CHAR *)NULL);
}

if (sqlca.sqlcode < 0)
{
   error_handler();
}
```

```
 /*
 ** Generated code ends here.
 */


}


void
cursor_sql()
{

 /*
 ** SQL STATEMENT: 15
 ** EXEC SQL BEGIN DECLARE SECTION;
 */

     char   ename[30], addr[30];
     int    ennum, mgr, dept;

 /*
 ** SQL STATEMENT: 16
 ** EXEC SQL END DECLARE SECTION;
 */

 /*
 ** Generated code ends here.
 */


 printf("beginning of sql cursor operations exercise\n");


 /*
 ** SQL STATEMENT: 17
 ** EXEC SQL DECLARE C1 CURSOR FOR Select * FROM employee WHERE name >=
:ename;

 **
 */


 strcpy(ename, "al");
```

```
/*
** SQL STATEMENT: 18
** EXEC SQL OPEN C1 USING :ename;
*/
_sqlhandles.connName.lnlen = CS_UNUSED;
_sqlhandles.curName.fnlen = 2;
strcpy(_sqlhandles.curName.first_name, "C1");
if ((_sqlretcode = _sqlprolog(&_sqlhandles, _SQL_CURSOR_TYPE, _SQL_CREATE,
   &sqlca, (CS_INT *)NULL, (CS_CHAR *)NULL)) == CS_SUCCEED)
{
   if (_sqlhandles.doDecl == CS_TRUE)
   {
      _sqlretcode = ct_cursor(_sqlhandles.cur.command, CS_CURSOR_DECLARE, ""
"C1", 2, "Select * FROM employee WHERE name >= @sql0_ename ", 49, CS_UNUSED);
      _sqldfmtCS_CHAR_TYPE.maxlength = 30;
      _sqldfmtCS_CHAR_TYPE.status = CS_INPUTVALUE;
      _sqlretcode = ct_param(_sqlhandles.cur.command,
         &_sqldfmtCS_CHAR_TYPE, NULL, 0, 0);
      _sqldfmtCS_CHAR_TYPE.status = 0;
      _sqldfmtCS_CHAR_TYPE.maxlength = 0;
   }

   _sqlretcode = ct_cursor(_sqlhandles.cur.command, CS_CURSOR_OPEN, NULL,
      CS_UNUSED, NULL, CS_UNUSED, CS_UNUSED);
   _sqldfmtCS_CHAR_TYPE.maxlength = 30;
   _sqldfmtCS_CHAR_TYPE.status = CS_INPUTVALUE;
   _sqlretcode = ct_param(_sqlhandles.cur.command, &_sqldfmtCS_CHAR_TYPE,
      ename, CS_NULLTERM, 0);
   _sqldfmtCS_CHAR_TYPE.status = 0;
   _sqldfmtCS_CHAR_TYPE.maxlength = 0;
   _sqlretcode = ct_send(_sqlhandles.cur.command);
   while ((_sqlretcode = ct_results(_sqlhandles.cur.command, &_sqlrestype))
      == CS_SUCCEED && _sqlrestype != CS_CURSOR_RESULT)
   {
      ;
   }

   _sqlretcode = _sqlepilog(&_sqlhandles, _SQL_CURSOR_TYPE, &sqlca, (CS_INT
      *)NULL, (CS_CHAR *)NULL);
   if (_sqlretcode != CS_FAIL)
   {
      _sqlhandles.bind = CS_TRUE;
      _sqlhandles.cur.buffer = &_sqlhandles.bind;
      _sqlhandles.cur.buflen = sizeof(_sqlhandles.bind);
      if (_sqlhandles.doDecl == CS_TRUE)
```

```
    {
      _sqlhandles.curName.thinkexists = CS_FALSE;
      }
       else {
      _sqlhandles.curName.thinkexists = CS_TRUE;
      }

      _sqlretcode = cs_objects(_sqlhandles.ctx, CS_SET,
        &_sqlhandles.curName, &_sqlhandles.cur);
  }

  if (_sqlretcode == CS_FAIL)
  {
    if ((_sqlretcode = cs_diag(_sqlhandles.ctx, CS_GET, SQLCA_TYPE, 1,
      &sqlca)) == CS_FAIL)
    {
      _sqlsetintrerr(&sqlca, (CS_INT *)NULL, (CS_CHAR *)NULL,
        (CS_INT)_SQL_INTRERR_25002);
    }

      }

  _sqlretcode = _sqlepilog(&_sqlhandles, _SQL_CURSOR_TYPE, &sqlca, (CS_INT
    *)NULL, (CS_CHAR *)NULL);
}

if (sqlca.sqlcode < 0)
{
  error_handler();
}


/*
** Generated code ends here.
*/


printf("employee name   \t number \t department \t manager \t address \n");
printf("--------------- \t ------ \t ---------- \t ------- \t ----------\n");
while (sqlca.sqlcode >= 0)
{

    /*
    ** SQL STATEMENT: 19
    ** EXEC SQL FETCH C1 INTO :ennum, :ename, :addr, :mgr, :dept;
    */
```

```
_sqlhandles.connName.lnlen = CS_UNUSED;
_sqlhandles.curName.fnlen = 2;
strcpy(_sqlhandles.curName.first_name, "C1");
if ((_sqlretcode = _sqlprolog(&_sqlhandles, _SQL_CURSOR_TYPE, _SQL_GET,
    &sqlca, (CS_INT *)NULL, (CS_CHAR *)NULL)) == CS_SUCCEED)
{
        _sqlretcode = ct_bind(_sqlhandles.cur.command, 1,
          &_sqldfmtCS_INT_TYPE, &ennum, NULL, NULL);
   _sqldfmtCS_CHAR_TYPE.maxlength = 30;
        _sqlretcode = ct_bind(_sqlhandles.cur.command, 2,
          &_sqldfmtCS_CHAR_TYPE, ename, NULL, NULL);
   _sqldfmtCS_CHAR_TYPE.maxlength = 0;
   _sqldfmtCS_CHAR_TYPE.maxlength = 30;
        _sqlretcode = ct_bind(_sqlhandles.cur.command, 3,
          &_sqldfmtCS_CHAR_TYPE, addr, NULL, NULL);
   _sqldfmtCS_CHAR_TYPE.maxlength = 0;
        _sqlretcode = ct_bind(_sqlhandles.cur.command, 4,
          &_sqldfmtCS_INT_TYPE, &mgr, NULL, NULL);
        _sqlretcode = ct_bind(_sqlhandles.cur.command, 5,
          &_sqldfmtCS_INT_TYPE, &dept, NULL, NULL);
   _sqlretcode = ct_fetch(_sqlhandles.cur.command, CS_UNUSED,
     CS_UNUSED, CS_UNUSED, &_sqlrowsread);
   if (_sqlretcode == CS_END_DATA)
   {
     while ((_sqlretcode = ct_results(_sqlhandles.cur.command,
       &_sqlrestype)) == CS_SUCCEED)
     {
       ;
     }

     if (_sqlretcode != CS_END_RESULTS)
     {
       _sqlretcode = ct_cancel(NULL, _sqlhandles.cur.command,
         CS_CANCEL_ALL);
     }

   }

   _sqlretcode = _sqlepilog(&_sqlhandles, _SQL_CURSOR_TYPE, &sqlca,
     (CS_INT *)NULL, (CS_CHAR *)NULL);
}

if (sqlca.sqlcode < 0)
{
   error_handler();
}
```

```
    /*
    ** Generated code ends here.
    */

    printf("%s\t %d \t %d \t %d \t %s\n\n", ename, ennum, dept, mgr, addr);

}


/*
** SQL STATEMENT: 20
** EXEC SQL CLOSE C1;
*/
_sqlhandles.connName.lnlen = CS_UNUSED;
_sqlhandles.curName.fnlen = 2;
strcpy(_sqlhandles.curName.first_name, "C1");
if ((_sqlretcode = _sqlprolog(&_sqlhandles, _SQL_CURSOR_TYPE, _SQL_GET,
   &sqlca, (CS_INT *)NULL, (CS_CHAR *)NULL)) == CS_SUCCEED)
{
   _sqlretcode = ct_cursor(_sqlhandles.cur.command, CS_CURSOR_CLOSE, NULL,
      CS_UNUSED, NULL, CS_UNUSED, CS_UNUSED);
   _sqlretcode = ct_send(_sqlhandles.cur.command);
   while ((_sqlretcode = ct_results(_sqlhandles.cur.command, &_sqlrestype))
      == CS_SUCCEED)
   {
      ;
   }

   if (_sqlretcode != CS_END_RESULTS)
   {
      _sqlretcode = ct_cancel(NULL, _sqlhandles.cur.command,
         CS_CANCEL_ALL);
   }
    else {
      _sqlretcode = CS_SUCCEED;
   }

   _sqlretcode = _sqlepilog(&_sqlhandles, _SQL_CURSOR_TYPE, &sqlca, (CS_INT
      *)NULL, (CS_CHAR *)NULL);
}

if (sqlca.sqlcode < 0)
{
   error_handler();
```

```
    }

    /*
    ** Generated code ends here.
    */

    printf("end of sql cursor operations exercise\n");

}
```

ISQL File

```
USE mydb
GO

CREATE PROCEDURE isql1_mytag;1
  AS
   INSERT INTO employee VALUES(1, 'hillary rodham clinton', 1)
GO

CREATE PROCEDURE isql1_mytag;2
  AS
   INSERT INTO employee VALUES(2, 'bill clinton', 1)
GO

CREATE PROCEDURE isql1_mytag;3
  AS
   INSERT INTO employee VALUES(3, 'al gore', 1)
GO

CREATE PROCEDURE isql1_mytag;4
  AS
   INSERT INTO department VALUES(1, 'white house', 1)
GO

CREATE PROCEDURE isql1_mytag;5
  AS
   COMMIT WORK
GO

CREATE PROCEDURE isql1_mytag;6
  AS
   COMMIT WORK
GO

CREATE PROCEDURE isql1_mytag;7
  AS
   COMMIT WORK
GO

CREATE PROCEDURE isql1_mytag;8
     (@sql0_mynum int, @sql1_myfloat real, @sql2_mydouble float)
  AS
   INSERT INTO dummytest VALUES (@sql0_mynum , @sql1_myfloat ,
     @sql2_mydouble )
```

```
GO

CREATE PROCEDURE isql1_mytag;9
      (@sql0_mynum int OUTPUT, @sql1_myfloat real OUTPUT,
       @sql2_mydouble float OUTPUT, @sql3_twonum int)
   AS
    SELECT * FROM dummytest WHERE mynum = @sql3_twonum
GO

CREATE PROCEDURE isql1_mytag;10
   AS
    ROLLBACK WORK
GO
```

Listing File

```
*******************************************************************************

  Sybase, Inc.
  Embedded SQL Precompiler

  Type and Version: cpre/10.0/B//Unknown/3/Tue Jul 27 16:25:39 PDT 1993
  Host Language Compiler: ANSI_C
  Date: Wed Jul 28 15:44:37 1993

  PRECOMPILER LISTING FILE

*******************************************************************************


1   #include <stdio.h>
2   #include <esql.h>
3
4   EXEC SQL BEGIN DECLARE SECTION;
5     char   server_name[30];
6     int    ennum;
7     int    ennum_type;
8     char   ename[30];
9     short  ename_indic;
10    char   addr[30];
11    int    dept;
12    int    mgr;
13    short  mgr_indic;
14    int    cnt;
15    int    occur;
16    int    descnt;
17    int    condcnt;
18    int    diag_cnt;
19    int    num_msgs;
20    char   user_id[30];
21    char   pass_id[30];
22    char   str1[256];
23    char   str2[256];
24    char   str3[256];
25    char   str4[256];
26  EXEC SQL END DECLARE SECTION;
27
28          long   sqlcode;
29
```

```
30
31  EXEC SQL WHENEVER SQLERROR CALL error_handler();
32
33  EXEC SQL INCLUDE SQLCA;
34
35  void
36  main()
37  {
38   strcpy(user_id, "sa");
39   strcpy(pass_id, "");
40   printf("enter in your server name: ");
41   scanf("%[^\n]", server_name);
42   EXEC SQL CONNECT :user_id IDENTIFIED BY :pass_id USING :server_name;
43
44   diag_cnt = 20;
45   EXEC SQL SET TRANSACTION DIAGNOSTICS SIZE :diag_cnt;
46
47   init_table();
48   static_sql();
49   dyn_m1();
50   dyn_m2();
51   dyn_m3();
52   dyn_m4();
53   cleanup();
54
55   EXEC SQL DISCONNECT ALL;
56
57  }
58
59
60

61  void
62  error_handler()
63  {
64   EXEC SQL GET DIAGNOSTICS :num_msgs = NUMBER;
65   for (condcnt=0; condcnt < num_msgs; condcnt++)
66   {
67        EXEC SQL GET DIAGNOSTICS EXCEPTION :condcnt
68                  :sqlca = SQLCA_INFO;
69        printf("sqlcode is: %d\n message text: %s\n", sqlca.sqlcode,
70             sqlca.sqlerrm.sqlerrmc);
71   }
72
73  }
74
```

```
...

116
117     dept = 44;
118     EXEC SQL UPDATE employee SET department = :dept
119         WHERED empnum = :ennum;
.............^
Sybase Server error. Server: Msgno: 102  Severity: 15 State: 1   Message: Incorrect syntax near
'whered'.
120
121     EXEC SQL DECLARE C1 CURSOR FOR Select * FROM employee WHERE name =
:ename;
.....................................................................^
Sybase server error.  Server:  Msgno: 137.  Severity: 15 State: 2   Message: Must declare variable
'@sql0_ename'.
...


Unable to find the SQL statement 'WHENEVER WARNING'.
Unable to find the SQL statement 'WHENEVER NOT FOUND'.
0 Error(s) and 12 Warning(s) found.
Statistical Report:
      Program name: cpre
      Options specified: -L
      Input file name: esql.pc
      Listing file name: esql.lst
      Target file name: esql.c
      ISQL file name:
      Tag ID specified:
      Compiler used: ANSI_C
      Open Client version: CS_VERSION_100
      Total number of messages: 24
      Number of information messages: 12
      Number of warning messages: 12
      Number of error messages: 0
      Number of SQL statements parsed: 95
      Number of host variables declared: 22
      Number of SQL cursors declared: 3
      Number of dynamic SQL statements: 28
      Connection(s) information:
            User id: sa
            Server: Dynamic10
            Database:
```

Dynamic SQL Example

```
/*
**    Sample program demonstration the use of Dynamic SQL's 4
**    methods
*/

extern void main();
extern void error_handler();
extern void dyn_sample();
extern int get_response();
extern void dyn_m1();
extern void dyn_m2a();
extern void dyn_m2b();
extern void dyn_m3();
extern void dyn_m4();
extern void fill_descriptor();
extern void print_descriptor();
extern void connection();
extern void disconnect();
extern void commit_work();
extern void rollback_work();

EXEC SQL WHENEVER SQLERROR CALL error_handler();
EXEC SQL INCLUDE SQLCA;

void
main()
{
  char user_id[128], pass_id[128], server_name[128];

  strcpy(user_id, "sa");
  strcpy(pass_id, "");
  connection(user_id, pass_id, server_name);
  dyn_sample();
  rollback_work();
  disconnect(server_name);

}

void
connection(user, password, server)
EXEC SQL BEGIN DECLARE SECTION;
  char *user;
  char *password;
```

```
  char *server;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL BEGIN DECLARE SECTION;
     char   database[128];
     int    diag_cnt;
  EXEC SQL END DECLARE SECTION;

  printf("enter in the server name: ");
  scanf("%s", server);
  printf("enter in the database name: ");
  scanf("%s", database);
  printf("enter in the desired number of diagnostics per statement: ");
  scanf("%d", &diag_cnt);

  EXEC SQL CONNECT :user IDENTIFIED BY :password USING :server;

  EXEC SQL USE :database;

  EXEC SQL SET TRANSACTION DIAGNOSTICS SIZE :diag_cnt;

}

void
disconnect(connect_name)
EXEC SQL BEGIN DECLARE SECTION;
  char *connect_name;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL DISCONNECT :connect_name;
}

void
commit_work()
{
  EXEC SQL COMMIT WORK;
}

void
rollback_work()
{
  EXEC SQL ROLLBACK WORK;
}

void
error_handler()
```

```
{
 EXEC SQL BEGIN DECLARE SECTION;
    long   num_msgs, condcnt;
 EXEC SQL END DECLARE SECTION;

 EXEC SQL GET DIAGNOSTICS :num_msgs = NUMBER;
 for (condcnt=0; condcnt < num_msgs; condcnt++)
 {
      EXEC SQL GET DIAGNOSTICS EXCEPTION :condcnt
                  :sqlca = SQLCA_INFO;
    printf("sqlcode is: %d\n message text: %s\n", sqlca.sqlcode,
         sqlca.sqlerrm.sqlerrmc);
 }

}



void
dyn_sample()
{
  int option;

  printf("\nDynamic SQL Sample Program.\n\n");
  while ((option = get_response()) != 0)
  {
     if (option == 1)
        dyn_m1();
     else if (option == 2)
        dyn_m2a();
     else if (option == 3)
        dyn_m2b();
     else if (option == 4)
        dyn_m3();
     else if (option == 5)
        dyn_m4();
     else
        printf("You have select an invalid option.  Please try again.\n");

  }
}

int
get_response()
{
  int option = -1, retcode;
```

```
  retcode = fflush(stdout);
  printf("\n0) Exit Dynamic SQL Sample Program\n");
  printf("1) Dynamic SQL Method 1\n");
  printf("2) Dynamic SQL Method 2 (non-Select)\n");
  printf("3) Dynamic SQL Method 2 (single-row Select)\n");
  printf("4) Dynamic SQL Method 3\n");
  printf("5) Dynamic SQL Method 4\n");
  printf("\nSelect one of the following Dynamic SQL Methods: ");
  scanf("%d", &option);
  return(option);

}



void
dyn_m1()
{
  EXEC SQL BEGIN DECLARE SECTION;
      char    str1[255];
  EXEC SQL END DECLARE SECTION;

  printf("\n\nDynamic SQL Method 1\n");
  printf("enter in a non-Select SQL statement: \n  ");
  scanf("%[^/]", str1);

  EXEC SQL EXECUTE IMMEDIATE :str1;

  strcpy(str1, "");
  printf("Dynamic SQL Method 1 completed\n\n");

}



void
dyn_m2a()
{
  EXEC SQL BEGIN DECLARE SECTION;
      int     occur;
      char    str2[255];
  EXEC SQL END DECLARE SECTION;
```

```
 printf("\n\nDynamic SQL Method 2 (non-Select)\n");
 printf("Enter in a statement to modify the date of a sales:\n  ");
 scanf("%[^/]", str2);
 printf("\nEnter in the number of ? in the SQL statement: ");
 scanf("%d", &occur);

 EXEC SQL PREPARE S1 FROM :str2;

 EXEC SQL ALLOCATE DESCRIPTOR DINOUT WITH MAX :occur;

 EXEC SQL DESCRIBE INPUT S1 USING SQL DESCRIPTOR DINOUT;

 fill_descriptor("DINOUT");

 EXEC SQL EXECUTE S1 USING SQL DESCRIPTOR DINOUT;

 EXEC SQL DEALLOCATE PREPARE S1;

 EXEC SQL DEALLOCATE DESCRIPTOR DINOUT;

 strcpy(str2, "");
 printf("Dynamic SQL Method 2 (non-Select) completed\n");

}



void dyn_m2b()
{
 EXEC SQL BEGIN DECLARE SECTION;
     int    occur;
     char   str2[255];
 EXEC SQL END DECLARE SECTION;

 printf("\n\nDynamic SQL Method 2 (single-row Select)\n");
 printf("Enter in a Select statement that will return a single row");
 printf(" for a certain author:\n  ");
 scanf("%[^/]", str2);
 printf("\nEnter in the larger of the columns to be retrieved or the number ");
 printf("of ? in the SQL statement: ");
 scanf("%d", &occur);

 EXEC SQL PREPARE S2 FROM :str2;

 EXEC SQL ALLOCATE DESCRIPTOR DINOUT WITH MAX :occur;
```

```
 EXEC SQL DESCRIBE INPUT S2 USING SQL DESCRIPTOR DINOUT;

 fill_descriptor("DINOUT");

 EXEC SQL EXECUTE S2 INTO SQL DESCRIPTOR DINOUT USING SQL DESCRIPTOR
DINOUT;

 print_descriptor("DINOUT");

 EXEC SQL DEALLOCATE PREPARE S2;

 EXEC SQL DEALLOCATE DESCRIPTOR DINOUT;

 strcpy(str2, "");
 printf("Dynamic SQL Method 2 (single-row select) completed\n");

}



void dyn_m3()
{
 EXEC SQL BEGIN DECLARE SECTION;
     char   str3[255], pub_id[4], pub_name[40], city[20], state[2];
 EXEC SQL END DECLARE SECTION;

 printf("\n\nDynamic SQL Method 3\n");
 printf("Enter in a Select statement to retrieve the list of publishers:\n  ");
 scanf("%[^/]", str3);

 EXEC SQL PREPARE S3 FROM :str3;

 EXEC SQL DECLARE C1 CURSOR FOR S3;

 EXEC SQL OPEN C1;

 printf("  id\tname\tcity\tstate\n");
 printf("-----------------------------------------------------------------");

 while (sqlca.sqlcode == 0)
 {
     EXEC SQL FETCH C1 INTO :pub_id, :pub_name, :city, :state;

     printf("  %s\t%s\t%s\t%s\n", pub_id, pub_name, city, state);
 }
```

```
 EXEC SQL CLOSE C2;

 EXEC SQL DEALLOCATE PREPARE S7;

 strcpy(str3, "");
 printf("Dynamic SQL Method 3 completed\n\n");

}



void dyn_m4()
{
 EXEC SQL BEGIN DECLARE SECTION;
     int    occur;
     char   str4[255];
 EXEC SQL END DECLARE SECTION;

 printf("\n\nDynamic sql Method 4\n");
 printf("Enter in a Select statement to retrieve any kind of ");
 printf("information from the pubs database:\n  ");
 scanf("%[^/]", str4);
 printf("\nEnter in the larger of the columns to be retrieved or the number ");
 printf("of ? in the SQL statement: ");
 scanf("%d", &occur);

 EXEC SQL PREPARE S4 FROM :str4;

 EXEC SQL DECLARE C2 CURSOR FOR S4;

 EXEC SQL DESCRIBE INPUT S4 USING SQL DESCRIPTOR DINOUT;

 fill_descriptor("DINOUT");

 EXEC SQL OPEN C2 USING SQL DESCRIPTOR DINOUT;

 EXEC SQL DESCRIBE OUTPUT S4 USING SQL DESCRIPTOR DINOUT;

 while (sqlca.sqlcode == 0)
 {
     EXEC SQL FETCH C2 INTO SQL DESCRIPTOR DINOUT;

     print_descriptor("DINOUT");
 }
```

```
 EXEC SQL CLOSE C2;

 EXEC SQL DEALLOCATE DESCRIPTOR DINOUT;

 EXEC SQL DEALLOCATE PREPARE S4;

 strcpy(str4, "");
 printf("Dynamic SQL Method 4 completed\n\n");

}



void
print_descriptor(desc_out)
EXEC SQL BEGIN DECLARE SECTION;
  char       *desc_out;
EXEC SQL END DECLARE SECTION;
{
 EXEC SQL BEGIN DECLARE SECTION;
     int    int_buff, index_colcnt, coltype, descnt;
     char   char_buff[255], void_buff[255], colname[255];
 EXEC SQL END DECLARE SECTION;

 EXEC SQL GET DESCRIPTOR :desc_out :descnt = COUNT;

 printf("Column name \t\tColumn data\n");
 printf("----------- \t\t----------------------------------------\n");

 for (index_colcnt = 1; index_colcnt <= descnt; index_colcnt++)
 { /* get each column attribute */
     EXEC SQL GET DESCRIPTOR :desc_out VALUE :index_colcnt :coltype = TYPE;

     switch(coltype)
     {
       case 1:    /* character type */
         EXEC SQL GET DESCRIPTOR :desc_out VALUE :index_colcnt
                         :colname = NAME, :char_buff = DATA;
         printf("%s \t\t %s\n", colname, char_buff);
         break;

       case 4:    /* integer type */
         EXEC SQL GET DESCRIPTOR :desc_out VALUE :index_colcnt
                         :colname = NAME, :int_buff = DATA;
```

```
            printf("%s \t\t %d\n", colname, int_buff);
            break;

        default:   /* other types */
            EXEC SQL GET DESCRIPTOR :desc_out VALUE :index_colcnt
                        :colname = NAME, :void_buff = DATA;
            printf("%s \t\t %s\n", colname, void_buff);
            break;
    }
 }

}



void
fill_descriptor(desc_in)
EXEC SQL BEGIN DECLARE SECTION;
  char      *desc_in;
EXEC SQL END DECLARE SECTION;
{
 EXEC SQL BEGIN DECLARE SECTION;
    int          cnt, coltype, descnt;
    CS_TINYINT     tinyint_buff;
    CS_SMALLINT    smallint_buff;
    CS_INT         int_buff;
    float          real_buff;
    double         float_buff, doubleprec_buff;
    CS_NUMERIC     numeric_buff;
    CS_DECIMAL     decimal_buff;
    CS_CHAR        char_buff[255];
    CS_VARCHAR     varchar_buff;
    CS_DATETIME    date_buff;
    CS_DATETIME4   date4_buff;
    CS_MONEY       money_buff;
    CS_MONEY4      money4_buff;
    CS_BIT         bit_buff;
    CS_TEXT        text_buff[255];
    CS_IMAGE       image_buff[255];
    CS_BINARY      binary_buff[255];
    CS_VARBINARY   varbinary_buff;
 EXEC SQL END DECLARE SECTION;

 EXEC SQL GET DESCRIPTOR :desc_in :descnt = COUNT;

 for (cnt = 1; cnt <= descnt; cnt++)
```

```
{
    printf("Enter in the data type of the %d ?: ", cnt);
    scanf("%d", &coltype);

    switch(coltype)
    {
      case 1:    /* character type */
        printf("Enter in the value for the character data type:\n  ");
        scanf("%[^/]", char_buff);
        EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = 1,
                        DATA = :char_buff;
        break;

      case 2:    /* numeric type */
        printf("Enter in the value for the numeric data type.\n");
        printf("precision: ");
        scanf("%c", &numeric_buff.precision);
        printf("scale: ");
        scanf("%c", &numeric_buff.scale);
        printf("data: ");
        scanf("%[^/]", numeric_buff.array);
        EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                        DATA = :numeric_buff;
        break;

      case 3:    /* decimal type */
        printf("Enter in the value for the decimal data type.\n");
        printf("precision: ");
        scanf("%c", &decimal_buff.precision);
        printf("scale: ");
        scanf("%c", &decimal_buff.scale);
        printf("data: ");
        scanf("%[^/]", decimal_buff.array);
        EXEC SQL SET DESCRIPTOR DINOUT VALUE :cnt TYPE = :coltype,
                        DATA = :decimal_buff;
        break;

      case 4:    /* integer type */
        printf("Enter in the value for the integer data type:\n  ");
        scanf("%d", &int_buff);
        EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                        DATA = :int_buff;
        break;

      case 5:    /* smallint (short) type */
        printf("Enter in the value for the smallint data type:\n  ");
```

```
   scanf("%hd", &smallint_buff);
   EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                    DATA = :smallint_buff;
   break;

case 6:    /* float type */
   printf("Enter in the value for the float data type:\n  ");
   scanf("%lf", &float_buff);
   EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                    DATA = :float_buff;
   break;

case 7:    /* real (float) type */
   printf("Enter in the value for the real data type:\n  ");
   scanf("%f", &real_buff);
   EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                    DATA = :real_buff;
   break;

case 8:    /* double precision (double) type */
   printf("Enter the value for the double precision datatype:\n ");
   scanf("%lf", &doubleprec_buff);
   EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                    DATA = :doubleprec_buff;
   break;

case 9:    /* date, time or timestamp type */
   printf("Enter in the value for the date time data type.\n");
   printf("day: ");
   scanf("%ld", &date_buff.dtdays);
   printf("time: ");
   scanf("%ld", &date_buff.dttime);
   EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                    DATA = :date_buff;
   break;

case 10:   /* interval */
   break;

case 12:   /* character varying type */
   printf("Enter in the value for the varchar data:\n  ");
   scanf("%[^/]", varchar_buff.str);
   varchar_buff.len = strlen(varchar_buff.str);
   EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                    DATA = :varchar_buff;
   break;
```

```
case 13:   /* enumerated type */
   break;

case 14:   /* bit type */
   printf("Enter in the value for the bit data type:\n  ");
   scanf("%c", &bit_buff);
   EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                  DATA = :bit_buff;
   break;

case 15:   /* bit varying */
   break;

case 16:   /* boolean */
   break;

case 17:   /* Abstract data types */
   break;


/* implementation defined types */
case -3:   /* text type */
   printf("Enter in the value for the text data type:\n  ");
   scanf("%[^/]", text_buff);
   EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                  DATA = :text_buff;
   break;

case -4:   /* image type */
   printf("Enter in the value for the image data type:\n  ");
   scanf("%[^/]", image_buff);
   EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                  DATA = :image_buff;
   break;

case -5:   /* binary type */
   printf("Enter in the value for the binary data type:\n  ");
   scanf("%[^/]", binary_buff);
   EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                  DATA = :binary_buff;
   break;

case -6:   /* binary varying type */
   printf("Enter in the value for the varbinary data type:\n  ");
   scanf("%[^/]", varbinary_buff.array);
```

```
            varbinary_buff.len = strlen((CS_CHAR *)varbinary_buff.array);
            EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                        DATA = :varbinary_buff;
            break;

        case -8:   /* tinyint type */
            printf("Enter in the value for the tinyint data type:\n  ");
            scanf("%c", &tinyint_buff);
            EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                        DATA = :tinyint_buff;
            break;

        case -9:   /* small date, time or timestamp type */
            printf("Enter in the value for the date time data type.\n");
            printf("day: ");
            scanf("%hd", &date4_buff.days);
            printf("minutes: ");
            scanf("%hd", &date4_buff.minutes);
            EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                        DATA = :date4_buff;
            break;

        case -10:  /* money type */
            printf("Enter in the value for the money data type:\n  ");
            scanf("%ld", &money_buff.mnyhigh);
            EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                        DATA = :money_buff;
            break;

        case -11:  /* small money type */
            printf("Enter in the value for the money data type:\n  ");
            scanf("%ld", &money4_buff.mny4);
            EXEC SQL SET DESCRIPTOR :desc_in VALUE :cnt TYPE = :coltype,
                        DATA = :money4_buff;
            break;

        default:
            printf("error!! non-supported column type. try again.\n");
            break;
    }
  }

}
```

Stored Procedure Example

```
#include <stdio.h>
#include <strings.h>

extern void main();
extern void handle_error();
extern void tsql_1();
extern void tsql_2();
extern void tsql_3();
extern void tsql_4();
extern void tsql_5();

EXEC SQL BEGIN DECLARE SECTION;
    char   my_default[80];
    int    diag_cnt, num_msgs, condcnt, retcode;
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;

EXEC SQL WHENEVER SQLERROR CALL handle_error();

void
main()
{
    EXEC SQL CONNECT "sa" IDENTIFIED BY "" USING "Dynamic10";

    diag_cnt = 20;
    EXEC SQL SET TRANSACTION DIAGNOSTICS SIZE :diag_cnt;

    EXEC SQL USE mydb;

    EXEC SQL SET CHAINED OFF;

    tsql_5();
    tsql_1();
    tsql_2();
    tsql_3();
    tsql_4();
}

void
handle_error()
{
    EXEC SQL GET DIAGNOSTICS :num_msgs = NUMBER;
```

```
    for (condcnt = 0; condcnt < num_msgs; condcnt++)
    {
      EXEC SQL GET DIAGNOSTICS EXCEPTION :condcnt
                :sqlca = SQLCA_INFO;

      printf("sqlcode is: %d\n message text: %s\n",
          sqlca.sqlcode, sqlca.sqlerrm.sqlerrmc);
    }
}

void
tsql_1()
{

    EXEC SQL INSERT INTO employee VALUES(1, 'hillary rodham clinton', 1);
    EXEC SQL INSERT INTO employee VALUES(2, 'bill clinton', 1);
    EXEC SQL INSERT INTO employee VALUES(3, 'al gore', 1);

    EXEC SQL INSERT INTO department VALUES(1, 'white house', 1);

    EXEC SQL COMMIT WORK;

    printf("at end of tsql_1\n");
}

void
tsql_2()
{

    EXEC SQL CREATE PROCEDURE P2(@numid int, @o1 int out, @o2 char(30) out,
                @o3 char(30) out, @o4 int output) AS
        SELECT @o1 = e.empnum, @o2 = e.empname,
            @o3 = d.deptname, @o4 = d.manager
            FROM employee e, department d
            WHERE e.deptnum = d.deptnum
              AND e.empnum = @numid;

    printf("at end of tsql_2\n");
}

void
tsql_3()
{
    EXEC SQL BEGIN DECLARE SECTION;
      int       empnum, manager;
      char      empname[30], deptname[30];
```

```
    EXEC SQL END DECLARE SECTION;

    empnum = 1;
    EXEC SQL EXEC :retcode = P2(@numid = :empnum,
                        @o1 = :empnum out,
                        @o2 = :empname output,
                        @o3 = :deptname output,
                        @o4 = :manager out);
    printf("\noutput from execution of stored procedure p2\n");
    printf("number: %d\nname: %s\ndepartment: %s\nmanager: %d\n\n",
        empnum, empname, deptname, manager);

    EXEC SQL COMMIT WORK;

    printf("at end of tsql_3\n");
}

void
tsql_4()
{

    EXEC SQL SET CHAINED OFF;

    EXEC SQL DROP TABLE employee;

    EXEC SQL DROP TABLE department;

    EXEC SQL COMMIT WORK;

    EXEC SQL DROP PROCEDURE P2;

    printf("at end of tsql_4\n");
}


void
tsql_5()
{
    EXEC SQL BEGIN DECLARE SECTION;
      int       mynum;
      int       twonum;
      float     myfloat;
      double    mydouble;
    EXEC SQL END DECLARE SECTION;

    mynum = 100;
```

```
    myfloat = 331.79;
    mydouble = 8.44e+11;
    EXEC SQL INSERT INTO dummytest VALUES (:mynum, :myfloat, :mydouble);

    twonum = 100;
    mynum = 0;
    myfloat = 0;
    mydouble = 0;
    EXEC SQL SELECT * INTO :mynum, :myfloat, :mydouble
            FROM dummytest WHERE mynum = :twonum;
    printf("\ntsql_5:  Selection results\n");
    printf("number: %d\nfloat: %f\ndouble: %e\ndouble(again): %f\n\n",
        mynum, myfloat, mydouble, mydouble);

    EXEC SQL ROLLBACK WORK;

    printf("at end of tsql_5\n");
}
```

Modular Programming Feature Example

```
/*
**    Sample program demonstration the use of Dynamic SQL's 4
**    methods
*/

extern void main();
extern void error_handler();
extern void dyn_sample();
extern int get_response();
extern void dyn_m1();
extern void dyn_m2a();
extern void dyn_m2b();
extern void dyn_m3();
extern void dyn_m4();
extern void fill_descriptor();
extern void print_descriptor();
extern void connection();
extern void disconnect();
extern void commit_work();
extern void rollback_work();
extern void dynprep();
extern void dyndprep();
extern void dyndescin();
extern void dyndescout();
extern void dynexecimm();
extern void dynexec();
extern void dynexecus();
extern void dynexecin();
extern void dynexecusin();
extern void dyndcurs();
extern void dynocurs();
extern void dynfcurs();
extern void dynccurs();
extern void dynadesc();
extern void dynddesc();
extern void dyngcdesc();
extern void dyngdesc();
extern void dynscdesc();
extern void dynsdesc();

EXEC SQL WHENEVER SQLERROR CALL error_handler();

EXEC SQL INCLUDE SQLCA;
```

```
#define DB_NUM_SIZE          77
#define DB_CHAR_SIZE         256
#define DB_LONGCHAR_SIZE     64000
#define DB_MEGACHAR_SIZE     2000000

typedef struct descriptor
{
  char      name[128];
  short     type;
  long      length;
  short     precision;
  short     scale;
  short     nullable;
  short     indicator;
  union
  {
     char  c[256];
     unsigned char n[DB_NUM_SIZE];
     unsigned char de[DB_NUM_SIZE];
     long  i;
     short  s;
     float  f;
     float  r;
     double  dp;
     char  cv[DB_CHAR_SIZE];
     char  bt;
     char  cl[BD_LONGCHAR_SIZE];
     char  ct[DB_MEGACHAR_SIZE];
     char  ci[DB_MEGACHAR_SIZE];
     unsigned char bi[256];
     unsigned char biv[DB_CHAR_SIZE];
     unsigned char bil[DB_LONGCHAR_SIZE];
     short  it;
     struct
     {
       long  days;
       long  time;
     } dt;
     struct
     {
       unsigned short  days4;
       unsigned short  minutes4;
     } dt4;
     struct
     {
```

```
        long mnyhigh;
        unsigned long mnylow;
      } m;
      struct
      {
        long  mny4;
      } m4;
  } data;
} DESC_ATTR;



void
main()
{
  char user_id[128], pass_id[128], server_name[128];

  strcpy(user_id, "sa");
  strcpy(pass_id, "");
  connection(user_id, pass_id, server_name);
  dyn_sample();
  rollback_work();
  disconnect(server_name);

}

void
connection(user, password, server)
EXEC SQL BEGIN DECLARE SECTION;
  char *user;
  char *password;
  char *server;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL BEGIN DECLARE SECTION;
     char   database[128];
     int    diag_cnt;
  EXEC SQL END DECLARE SECTION;

  printf("enter in the server name: ");
  scanf("%s", server);
  printf("enter in the database name: ");
  scanf("%s", database);
  printf("enter in the desired number of diagnostics per statement: ");
  scanf("%d", &diag_cnt);
```

```
  EXEC SQL CONNECT :user IDENTIFIED BY :password USING :server;

  EXEC SQL USE :database;

  EXEC SQL SET TRANSACTION DIAGNOSTICS SIZE :diag_cnt;

}

void
disconnect(connect_name)
EXEC SQL BEGIN DECLARE SECTION;
  char *connect_name;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL DISCONNECT :connect_name;
}

void
commit_work()
{
  EXEC SQL COMMIT WORK;
}

void
rollback_work()
{
  EXEC SQL ROLLBACK WORK;
}

void
error_handler()
{
 EXEC SQL BEGIN DECLARE SECTION;
     long   num_msgs, condcnt;
 EXEC SQL END DECLARE SECTION;

 EXEC SQL GET DIAGNOSTICS :num_msgs = NUMBER;
 for (condcnt=0; condcnt < num_msgs; condcnt++)
 {
      EXEC SQL GET DIAGNOSTICS EXCEPTION :condcnt
                    :sqlca = SQLCA_INFO;
    printf("sqlcode is: %d\n message text: %s\n", sqlca.sqlcode,
         sqlca.sqlerrm.sqlerrmc);
 }
```

```
}

void
dyn_sample()
{
  int option;

  printf("\nDynamic SQL Sample Program.\n\n");
  while ((option = get_response()) != 0)
  {
     if (option == 1)
        dyn_m1();
     else if (option == 2)
        dyn_m2a();
     else if (option == 3)
        dyn_m2b();
     else if (option == 4)
        dyn_m3();
     else if (option == 5)
        dyn_m4();
     else
        printf("You have select an invalid option.  Please try again.\n");

  }
}

int
get_response()
{
  int option = -1;

  printf("\n0) Exit Dynamic SQL Sample Program\n");
  printf("1) Dynamic SQL Method 1\n");
  printf("2) Dynamic SQL Method 2 (non-Select)\n");
  printf("3) Dynamic SQL Method 2 (single-row Select)\n");
  printf("4) Dynamic SQL Method 3\n");
  printf("5) Dynamic SQL Method 4\n");
  printf("\nSelect one of the following Dynamic SQL Methods: ");
  scanf("%d", &option);
  return(option);

}


void
dyn_m1()
```

```
{
  char str[255];

  printf("\n\nDynamic SQL Method 1\n");
  printf("enter in a non-Select SQL statement: \n  ");
  scanf("%[^/]", str);

  dynexecimm(str);

  printf("Dynamic SQL Method 1 completed\n\n");

}

void
dyn_m2a()
{
  int   occur;
  char  str[256], stmt_id[128], desc_id[128];
  DESC_ATTR *   desc_attr;

  printf("\n\nDynamic SQL Method 2 (non-Select)\n");
  printf("Enter in a statement to modify the date of a sales:\n  ");
  scanf("%[^/]", str);
  printf("\nEnter in the number of ? in the SQL statement: ");
  scanf("%d", &occur);
  strcpy(stmt_id, "S2A");
  strcpy(desc_id, "DINOUT");

  dynprep(stmt_id, str);
  dynadesc(desc_id, occur);
  dyndescin(stmt_id, desc_id);
  fill_descriptor(desc_attr);
  dynsdesc(desc_id, desc_attr);
  dynexecus(stmt_id, desc_id);
  dyndprep(stmt_id);
  dynddesc(desc_id);

  printf("Dynamic SQL Method 2 (non-Select) completed\n");

}

void dyn_m2b()
{
  int   occur;
  char  str[255], stmt_id[128], desc_id[128];
  DESC_ATTR *   desc_attr;
```

```
 printf("\n\nDynamic SQL Method 2 (single-row Select)\n");
 printf("Enter in a Select statement that will return a single row");
 printf(" for a certain author:\n  ");
 scanf("%[^/]", str);
 printf("\nEnter in the larger of the columns to be retrieved or the number ");
 printf("of ? in the SQL statement: ");
 scanf("%d", &occur);
 strcpy(stmt_id, "S2B");
 strcpy(desc_id, "DINOUT");

 dynprep(stmt_id, str);
 dynadesc(desc_id, occur);
 dyndescin(stmt_id, desc_id);
 fill_descriptor(desc_attr);
 dynsdesc(desc_id, desc_attr);
 dynexecusin(stmt_id, desc_id, desc_id);
 dyngdesc(desc_id, desc_attr);
 print_descriptor(desc_attr);
 dyndprep(stmt_id);
 dynddesc(desc_id);

 printf("Dynamic SQL Method 2 (single-row select) completed\n");

}

void dyn_m3()
{
 char  str[256], stmt_id[128];
 EXEC SQL BEGIN DECLARE SECTION;
     char   cursor_id[128], pub_id[4], pub_name[40], city[20], state[2];
 EXEC SQL END DECLARE SECTION;

 printf("\n\nDynamic SQL Method 3\n");
 printf("Enter in a Select statement to retrieve the list of publishers:\n  ");
 scanf("%[^/]", str);
 strcpy(cursor_id, "C1");
 strcpy(stmt_id, "S3");

 dynprep(stmt_id, str);
 dyndcurs(cursor_id, stmt_id);

 EXEC SQL OPEN :cursor_id;

 printf("  id\tname\tcity\tstate\n");
 printf("-------------------------------------------------------------");
```

```
 while (sqlca.sqlcode == 0)
 {
     EXEC SQL FETCH :cursor_id INTO :pub_id, :pub_name, :city, :state;

     printf("  %s\t%s\t%s\t%s\n", pub_id, pub_name, city, state);
 }

 dynccur(cursor_id);
 dyndprep(stmt_id);

 printf("Dynamic SQL Method 3 completed\n\n");

}

void dyn_m4()
{
 int  occur;
 char  str[256], stmt_id[128], desc_id[128], cursor_id[128];

 printf("\n\nDynamic sql Method 4\n");
 printf("Enter in a Select statement to retrieve any kind of ");
 printf("information from the pubs database:\n  ");
 scanf("%[^/]", str);
 printf("\nEnter in the larger of the columns to be retrieved or the number ");
 printf("of ? in the SQL statement: ");
 scanf("%d", &occur);
 strcpy(stmt_id, "S4");
 strcpy(desc_id, "DINOUT");
 strcpy(cursor_id, "C2");

 dynprep(stmt_id, str);
 dyndcurs(cursor_id, stmt_id);
 dynadesc(desc_id, occur);
 dyndescin(stmt_id, desc_id);
 fill_descriptor(desc_attr);
 dynocurs(cursor_id, cursor_rows, desc_id);
 dyndescout(stmt_id, desc_id);

 while (sqlca.sqlcode == 0)
 {
     dynfcurs(cursor_id, desc_id);
     dyngdesc(desc_id, desc_attr);
     print_descriptor(desc_attr);
 }
```

```
 dynccurs(cursor_id);
 dyndprep(stmt_id);
 dynddesc(desc_id);

 printf("Dynamic SQL Method 4 completed\n\n");

}


void
print_descriptor(desc_attr)
 DESC_ATTR *  desc_attr;
{
 DESC_ATTR *  curr_attr;

 printf("name\ttype\tlength\tprec\tscale\tnull\indic\tColumn data\n");
 printf("-----------------------------------------------------------------\n");
 for (curr_attr = desc_attr; curr_attr != (DESC_ATTR *)NULL;
     curr_attr = curr_attr->next)
 {
    printf("%s\t%d\t", curr_attr->name, curr_attr->type);

    if (curr_attr->type == 1 || curr_attr_type == 12 || curr_attr->type == 14
       || (curr_attr->type <= -2 && curr_attr_type >= -7))
      printf("%d\t-\t-\t", curr_attr->length);
    else if (curr_attr->type == 2 || curr_attr_type == 3)
      printf("-\t%d\t%d\t", curr_attr->precision, curr_attr->scale);
    else if (curr_attr->type == 4 || curr_attr_type == 5 ||
          curr_attr->type == -8)
      printf("-\t-\t-\t");
    else if (curr_attr->type == 9 || curr_attr->type == -9)
      printf("-\t-\t-\t");
    else if (curr_attr->type == -10 || curr_attr->type == -11)
      printf("-\t-\t-\t");
    else if (curr_attr->type >= 6 && curr_attr_type <= 8)
      printf("-\t%d\t-\t", curr_attr->precision);

    printf("%d\t%d\n\t", curr_attr->nullable, curr_attr->indicator);
    switch(curr_attr->type)
    {
     case 1:
       printf("%s\n", curr_attr->data.c);
       break;
     case 2:
       printf("%s\n", curr_attr->data.n);
       break;
```

```
case 3:
  printf("%s\n", curr_attr->data.de);
  break;
case 4:
  printf("%d\n", curr_attr->data.i);
  break;
case 5:
  printf("%hd\n", curr_attr->data.s);
  break;
case 6:
  printf("%f\n", curr_attr->data.f);
  break;
case 7:
  printf("%f\n", curr_attr->data.r);
  break;
case 8:
  printf("%lf\n", curr_attr->data.do);
  break;
case 9:
  printf("%ld %ld\n",
      curr_attr->data.dt.days, curr_attr->data.dt.time);
  break;
case 10:
  break;
case 12:
  printf("%s\n", curr_attr->data.cv);
  break;
case 13:
  break;
case 14:
  printf("%c\n", curr_attr->data.bt);
  break;
case 15:
  break;
case 16:
  break;
case 17:
  break;
case -2:
  printf("%s\n", curr_attr->data.cl);
  break;
case -3:
  printf("%s\n", curr_attr->data.ct);
  break;
case -4:
  printf("%s\n", curr_attr->data.ci);
```

```
        break;
      case -5:
        printf("%s\n", curr_attr->data.bi);
        break;
      case -6:
        printf("%s\n", curr_attr->data.biv);
        break;
      case -7:
        printf("%s\n", curr_attr->data.bil);
        break;
      case -8:
        printf("%hd\n", curr_attr->data.it);
        break;
      case -9:
        printf("%hd %hd\n",
             curr_attr->data.dt4.days4, curr_attr->data.dt4.minutes4);
        break;
      case -10:
        printf("%ld %ld\n",
             curr_attr->data.m.mnyhigh, curr_attr->data.m.mnylow);
        break;
      case -11:
        printf("%ld\n", curr_attr->data.m4.mny4);
        break;
      default:
        break;
    } /* end of switch */
  }

}

void
fill_descriptor(desc_attr)
  DESC_ATTR *   desc_attr;
{
  DESC_ATTR *  curr_attr;
  short        desc_cnt, param_index;

  printf("how many dynamic parameters will you fill in: );
  scanf("%d", &desc_cnt);

  for (param_index = 1;  param_index <= desc_cnt; param_index++)
  {
     if (curr_attr == (DESC_ATTR *)NULL)
     {
        curr_attr = (DESC_ATTR *)malloc(sizeof(DESC_ATTR));
```

```
    desc_attr = curr_attr;
  }
  else
  {
    curr_attr->next = (DESC_ATTR *)malloc(sizeof(DESC_ATTR));
    curr_attr = curr_attr->next;
  }
  curr_attr->next = (DESC_ATTR *)NULL;

  printf("enter in the name of the parameter: ");
  scanf("%[^/]", curr_attr->name);
  printf("enter in the type: ");
  scanf("%d", &curr_attr->type);

  if (curr_attr->type == 1 || curr_attr_type == 12 ||
    curr_attr->type == 14 ||
    (curr_attr->type <= -2 && curr_attr_type >= -7))
  {
    printf("enter in the length: ");
    scanf("%d", &curr_attr->length);
    curr_attr->precision = -1;
    curr_attr->scale = -1;
  }
  else if (curr_attr->type == 2 || curr_attr_type == 3)
  {
    curr_attr->length = -1;
    printf("enter in the precision: ");
    scanf("%d", &curr_attr->precision);
    printf("enter in the scale: ");
    scanf("%d", &curr_attr->scale);
  }
  else if (curr_attr->type == 4 || curr_attr_type == 5 ||
      curr_attr->type == -8)
  {
    curr_attr->length = -1;
    curr_attr->precision = -1;
    curr_attr->scale = -1;
  }
  else if (curr_attr->type >= 6 && curr_attr_type <= 8)
  {
    curr_attr->length = -1;
    printf("enter in the precision: ");
    scanf("%d", &curr_attr->precision);
    curr_attr->scale = 0;
  }
  else if (curr_attr->type == 9 || curr_attr->type == -9)
```

```
{
   curr_attr->length = -1;
   curr_attr->precision = -1;
   curr_attr->scale = -1;
}
else if (curr_attr->type == -10 || curr_attr->type == -11)
{
   curr_attr->length = -1;
   curr_attr->precision = -1;
   curr_attr->scale = -1;
}
else
{
   curr_attr->length = -1;
   curr_attr->precision = -1;
   curr_attr->scale = -1;
}

printf("enter in the nullable value (0 or 1): ");
scanf("%d", &curr_attr->nullable);
printf("enter in the indicator value: ");
scanf("%d", &curr_attr->indicator);

printf("enter in the data: ");
switch(curr_attr->type)
{
case 1:
  scanf("%[^/]", curr_attr->data.c);
  break;
case 2:
  scanf("%[^/]", curr_attr->data.n);
  break;
case 3:
  scanf("%[^/]", curr_attr->data.de);
  break;
case 4:
  scanf("%d", &curr_attr->data.i);
  break;
case 5:
  scanf("%hd", &curr_attr->data.s);
  break;
case 6:
  scanf("%f", &curr_attr->data.f);
  break;
case 7:
  scanf("%f", &curr_attr->data.r);
```

```
    break;
  case 8:
    scanf("%lf", &curr_attr->data.do);
    break;
  case 9:
    scanf("%ld %ld", &curr_attr->data.dt.days, &curr_attr->data.dt.time);
    break;
  case 10:
    break;
  case 12:
    scanf("%[^/]", curr_attr->data.cv);
    break;
  case 13:
    break;
  case 14:
    scanf("%c", &curr_attr->data.bt);
    break;
  case 15:
    break;
  case 16:
    break;
  case 17:
    break;
  case -2:
    scanf("%[^/]", curr_attr->data.cl);
    break;
  case -3:
    scanf("%[^/]", curr_attr->data.ct);
    break;
  case -4:
    scanf("%[^/]", curr_attr->data.ci);
    break;
  case -5:
    scanf("%[^/]", curr_attr->data.bi);
    break;
  case -6:
    scanf("%[^/]", curr_attr->data.biv);
    break;
  case -7:
    scanf("%[^/]", curr_attr->data.bil);
    break;
  case -8:
    scanf("%hd", &curr_attr->data.it);
    break;
  case -9:
    scanf("%ld %ld",
```

```
            &curr_attr->data.dt4.days4, &curr_attr->data.dt4.minutes4);
        break;
      case -10:
        scanf("%ld %ld",
            &curr_attr->data.m.mnyhigh, &curr_attr->data.m.mnylow);
        break;
      case -11:
        scanf("%ld", &curr_attr->data.m4.mny4);
        break;
      default:
        break;
      } /* end of switch */
      printf("\n\n");
  }

}


/*
** dynamic statement operations
*/
void
dynprep(stmt_id, str)
EXEC SQL BEGIN DECLARE SECTION;
  char *      stmt_id;
  char *      str;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL PREPARSE :stmt_id FROM :str;
}

void dyndprep(stmt_id)
EXEC SQL BEGIN DECLARE SECTION;
  char *      stmt_id;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL DEALLOCATE PREPARE :stmt_id;
}

void
dyndescin(stmt_id, desc_id)
EXEC SQL BEGIN DECLARE SECTION;
  char *      stmt_id;
  char *      desc_id;
EXEC SQL END DECLARE SECTION;
{
```

```
   EXEC SQL DESCRIBE INPUT :stmt_id USING SQL DESCRIPTOR :desc_id;
}

void
dyndescout(stmt_id, desc_id)
EXEC SQL BEGIN DECLARE SECTION;
  char *      stmt_id;
  char *      desc_id;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL DESCRIBE OUTPUT :stmt_id USING SQL DESCRIPTOR :desc_id;
}

void
dynexecimm(str)
EXEC SQL BEGIN DECLARE SECTION;
  char *      str;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL EXECUTE IMMEDIATE :str;
}

void
dynexec(stmt_id)
EXEC SQL BEGIN DECLARE SECTION;
  char *      stmt_id;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL EXECUTE :stmt_id;
}

void
dynexecus(stmt_id, desc_id)
EXEC SQL BEGIN DECLARE SECTION;
  char *      stmt_id;
  char *      desc_id;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL EXECUTE :stmt_id USING SQL DESCRIPTOR :desc_id;
}

void
dynexecin(stmt_id, desc_id)
EXEC SQL BEGIN DECLARE SECTION;
  char *      stmt_id;
  char *      desc_id;
```

```
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL EXECUTE :stmt_id INTO SQL DESCRIPTOR :desc_id;
}

void
dynexecusin(stmt_id, desc_in, desc_out)
EXEC SQL BEGIN DECLARE SECTION;
  char *      stmt_id;
  char *      desc_in;
  char *      desc_out;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL EXECUTE :stmt_id USING SQL DESCRIPTOR :desc_in
                    INTO SQL DESCRIPTOR :desc_out;
}

/*
** cursor operations
*/
void
dyndcurs(cursor_id, stmt_id)
EXEC SQL BEGIN DECLARE SECTION;
  char *      cursor_id;
  char *      stmt_id;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL DECLARE :cursor_id CURSOR FOR :stmt_id;
}

void
dynocurs(cursor_id, cursor_rows, desc_id)
EXEC SQL BEGIN DECLARE SECTION;
  char *      cursor_id;
  long        cursor_rows;
  char *      desc_id;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL OPEN :cursor_id ROW_COUNT = :cursor_rows
                    USING SQL DESCRIPTOR :desc_id;
}

void
dynfcurs(cursor_id, desc_id)
EXEC SQL BEGIN DECLARE SECTION;
  char *      cursor_id;
```

```
  char *      desc_id;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL FETCH :cursor_id INTO SQL DESCRIPTOR :desc_id;
}

void
dynccurs(cursor_id)
EXEC SQL BEGIN DECLARE SECTION;
  char *      cursor_id;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL CLOSE :cursor_id;
}

/*
** descriptor operations
*/

void
dynadesc(desc_id, occurances)
EXEC SQL BEGIN DECLARE SECTION;
  char *      desc_id;
  short       occurances;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL ALLOCATE DESCRIPTOR :desc_id WITH MAX :occurances;
}

void
dynddesc(desc_id)
EXEC SQL BEGIN DECLARE SECTION;
  char *      desc_id;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL DEALLOCATE DESCRIPTOR :desc_id;
}

void
dyngcdesc(desc_id, count)
EXEC SQL BEGIN DECLARE SECTION;
  char *      desc_id;
  short *     count;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL GET DESCRIPTOR :desc_id :count = COUNT;
```

```
}

void
dynscdesc(desc_id, count)
EXEC SQL BEGIN DECLARE SECTION;
  char *      desc_id;
  short       count;
EXEC SQL END DECLARE SECTION;
{
  EXEC SQL SET DESCRIPTOR :desc_id COUNT = :count;
}


/*
** dyngdesc
**
** Purpose:
**      get the attribute information from the SQL Descriptor
**
** Parameter(s):
**      desc_id: the name of the descriptor
**      desc_attr: the attribute information obtained from the descriptor
**
*/
void
dyngdesc(desc_id, desc_attr)
EXEC SQL BEGIN DECLARE SECTION;
  char *      desc_id;
  DESC_ATTR *  desc_attr;
EXEC SQL END DECLARE SECTION;
{
   EXEC SQL BEGIN DECLARE SECTION;
      short   desc_cnt, col_index, col_type;
      DESC_ATTR *     curr_attr;
   EXEC SQL END DECLARE SECTION;


   EXEC SQL GET DESCRIPTOR :desc_id :desc_cnt = COUNT;

   for (col_index = 1; col_index <= desc_cnt && sqlca.sqlcode == 0;
       col_index++)
   {
      if (curr_attr == (DESC_ATTR *)NULL)
      {
         curr_attr = (DESC_ATTR *)malloc(sizeof(DESC_ATTR));
         desc_attr = curr_attr;
```

```
    curr_attr->next = (DESC_ATTR *)NULL;
}
else
{
    curr_attr->next = (DESC_ATTR *)malloc(sizeof(DESC_ATTR));
    curr_attr = curr_attr->next;
    curr_attr->next = (DESC_ATTR *)NULL;
}

EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                        :curr_attr->type = TYPE,
                        :curr_attr->name = NAME,
                        :curr_attr->null = NULLABLE,
                        :curr_attr->indic = INDICATOR;

switch(curr_attr->type)
{
    case 1:    /* character type */
        EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                        :curr_attr->length = LENGTH,
                        :curr_attr->data.c = DATA;
        break;
    case 2:    /* numeric type */
        EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                        :curr_attr->prec = PRECISION,
                        :curr_attr->scale = SCALE,
                        :curr_attr->data.n = DATA;
        break;
    case 3:    /* decimal type */
        EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                        :curr_attr->prec = PRECISION,
                        :curr_attr->scale = SCALE,
                        :curr_attr->data.de = DATA;
        break;
    case 4:    /* integer type */
        EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                        :curr_attr->data.i = DATA;
        break;
    case 5:    /* smallint type */
        EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                        :curr_attr->data.s = DATA;
        break;
    case 6:    /* float type */
        EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                        :curr_attr->prec = PRECISION,
                        :curr_attr->data.f = DATA;
```

```
    break;
case 7:    /* real type */
    EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                    :curr_attr->prec = PRECISION,
                    :curr_attr->data.r = DATA;
    break;
case 8:    /* double precision type */
    EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                    :curr_attr->prec = PRECISION,
                    :curr_attr->data.do = DATA;
    break;
case 9:    /* date, time, timestamp type */
    EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                    :curr_attr->data.dt = DATA;
    break;
case 10:   /* interval type -- not supported yet */
    break;
case 12:   /* character varying type */
    EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                    :curr_attr->length = LENGTH,
                    :curr_attr->data.cv = DATA;
    break;
case 13:   /* enumerated type -- not supported yet */
    break;
case 14:   /* bit type */
    EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                    :curr_attr->length = LENGTH,
                    :curr_attr->data.bt = DATA;
    break;
case 15:   /* bit varying type -- not supported yet */
    break;
case 16:   /* boolean type -- not supported yet */
    break;
case 17:   /* abstract data type -- not supported yet */
    break;

/*
** implementation defined types
*/
case -2:   /* long char type */
    EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                    :curr_attr->length = LENGTH,
                    :curr_attr->data.cl = DATA;
    break;
case -3:   /* text type */
    EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
```

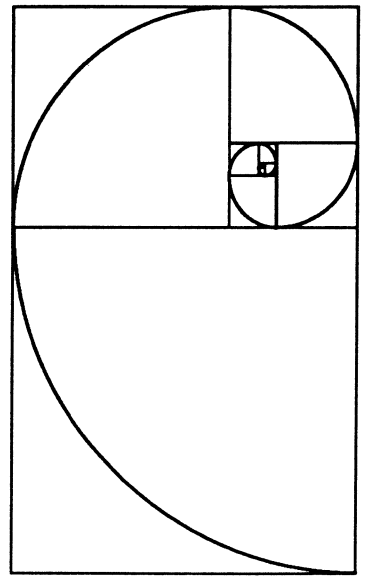
```
                    :curr_attr->length = LENGTH,
                    :curr_attr->data.ct = DATA;
      break;
    case -4:   /* image type */
      EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                    :curr_attr->length = LENGTH,
                    :curr_attr->data.ci = DATA;
      break;
    case -5:   /* binary type */
      EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                    :curr_attr->length = LENGTH,
                    :curr_attr->data.bi = DATA;
      break;
    case -6:   /* binary varying type */
      EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                    :curr_attr->length = LENGTH,
                    :curr_attr->data.biv = DATA;
      break;
    case -7:   /* long binary type */
      EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                    :curr_attr->length = LENGTH,
                    :curr_attr->data.bil = DATA;
      break;
    case -8:   /* tinyint type */
      EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                    :curr_attr->data.ti = DATA;
      break;
    case -9:   /* small date, time or timestamp type */
      EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                    :curr_attr->data.dt4 = DATA;
      break;
    case -10:  /* money type */
      EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                    :curr_attr->prec = PRECISION,
                    :curr_attr->scale = SCALE,
                    :curr_attr->data.m = DATA;
      break;
    case -11:  /* money type */
      EXEC SQL GET DESCRIPTOR :desc_id VALUE :col_index
                    :curr_attr->prec = PRECISION,
                    :curr_attr->scale = SCALE,
                    :curr_attr->data.m4 = DATA;
      break;
    default:
      break;
} /* end of switch */
```

```
    }

} /* dyngdesc() */


/*
** dynsdesc
**
** Purpose:
**      sets the attribute values of a descriptor
**
** Parameter(s):
**      desc_id: the name of the descriptor
**      desc_attr: the attribute values for the dynamic parameters
**
*/
void
dynsdesc(desc_id, desc_attr)
EXEC SQL BEGIN DECLARE SECTION;
  char *       desc_id;
  DESC_ATTR *  desc_attr;
EXEC SQL END DECLARE SECTION;
{
    EXEC SQL BEGIN DECLARE SECTION;
        short   col_index, desc_cnt;
        DESC_ATTR *     curr_attr;
    EXEC SQL END DECLARE SECTION;


    EXEC SQL GET DESCRIPTOR :desc_id :desc_cnt = COUNT;

    for (col_index = 1, curr_attr = desc_attr;
        col_index <= desc_cnt && sqlca.sqlcode == 0 &&
        curr_attr != (DESC_ATTR *)NULL;
        col_index++, curr_attr = curr_attr->next)
    {
        EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                                TYPE = :curr_attr->type,
                                NAME = :curr_attr->name,
                                NULLABLE = :curr_attr->null,
                                INDICATOR = :curr_attr->indic;

        switch(curr_attr->type)
        {
            case 1:    /* character type */
                EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
```

```
                LENGTH = :curr_attr->length,
                DATA = :curr_attr->data.c;
  break;
case 2:   /* numeric type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                PRECISION = :curr_attr->prec,
                SCALE = :curr_attr->scale,
                DATA = :curr_attr->data.n;
  break;
case 3:   /* decimal type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                PRECISION = :curr_attr->prec,
                SCALE = :curr_attr->scale,
                DATA = :curr_attr->data.de;
  break;
case 4:   /* integer type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                DATA = :curr_attr->data.i;
  break;
case 5:   /* smallint type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                DATA = :curr_attr->data.si;
  break;
case 6:   /* float type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                PRECISION = :curr_attr->prec,
                DATA = :curr_attr->data.f;
  break;
case 7:   /* real type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                PRECISION = :curr_attr->prec,
                DATA = :curr_attr->data.r;
  break;
case 8:   /* double precision type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                PRECISION = :curr_attr->prec,
                DATA = :curr_attr->data.do;
  break;
case 9:   /* date, time, timestamp type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                DATA = :curr_attr->data.dt;
  break;
case 10:  /* interval type -- not supported yet */
  break;
case 12:  /* character varying type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
```

```
                    LENGTH = :curr_attr->length,
                    DATA = :curr_attr->data.cv;
  break;
case 13:  /* enumerated type -- not supported yet */
  break;
case 14:  /* bit type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                    LENGTH = :curr_attr->length,
                    DATA = :curr_attr->data.bt;
  break;
case 15:  /* bit varying type -- not supported yet */
  break;
case 16:  /* boolean type -- not supported yet */
  break;
case 17:  /* abstract data type -- not supported yet */
  break;

/*
** implementation defined types
*/
case -2:  /* long char type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                    LENGTH = :curr_attr->length,
                    DATA = :curr_attr->data.cl;
  break;
case -3:  /* text type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                    LENGTH = :curr_attr->length,
                    DATA = :curr_attr->data.ct;
  break;
case -4:  /* image type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                    LENGTH = :curr_attr->length,
                    DATA = :curr_attr->data.ci;
  break;
case -5:  /* binary type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                    LENGTH = :curr_attr->length,
                    DATA = :curr_attr->data.bi;
  break;
case -6:  /* binary varying type */
  EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                    LENGTH = :curr_attr->length,
                    DATA = :curr_attr->data.biv;
  break;
case -7:  /* long binary type */
```

```
            EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                        LENGTH = :curr_attr->length,
                        DATA = :curr_attr->data.bil;
            break;
         case -8:   /* tinyint type */
            EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                        DATA = :curr_attr->data.ti;
            break;
         case -9:   /* small date, time or timestamp type */
            EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                        DATA = :curr_attr->data.dt4;
            break;
         case -10:  /* money type */
            EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                        PRECISION = :curr_attr->prec,
                        SCALE = :curr_attr->scale,
                        DATA = :curr_attr->data.m;
            break;
         case -11:  /* small money type */
            EXEC SQL SET DESCRIPTOR :desc_id VALUE :col_index
                        PRECISION = :curr_attr->prec,
                        SCALE = :curr_attr->scale,
                        DATA = :curr_attr->data.m4;
            break;
         default:
            break;

      } /* end of switch */
   }

} /* dynsdesc() */
```



Sybase

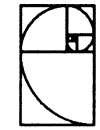
System 10 SQL Server

New Application Support Features

Author: Manish Chandra

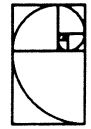
Application Development Features

- @@transtate
- ROLLBACK TRIGGER
- Extended Error Data (available via Open Client)
- Trigger Self-recursion
- Row Identity



@@transtate

- Applications need to know if a command such as "insert" has succeeded or failed
- An "insert" may trigger a set of events, which may
  - complete.
  - complete with errors.
  - be rolled back.
- Right now, it is difficult for an application to know if the insert succeeded
- @@transtate lets the application test if the insert or other command succeeded



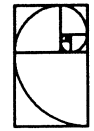
@@transtate: Old and New

- Old Method: Iterate the Fatal Errors

```
insert A
if (@@error in (< list of fatal errors >)
     /* The list of fatal error is long */
     print "Please resubmit after fixing errors."
else
     print "The record has been inserted."
```

- New Method: @@transtate

```
insert A
if (@@transtate = 1)     /*Succeeded*/
     print "The record has been inserted."
else
     print "Please resubmit after fixing errors."
```

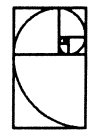


@@transtate

- The values for @@transtate are:
  - Transaction in progress 0
  - Transaction succeeded 1
  - Statement aborted 2
  - Transaction aborted 3

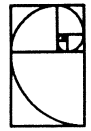
- @@transtate works correctly with triggers e.g:

  insert table A values (....)
  which triggers other events
  select @@transtate
  will reflect whether the insert into table A worked or got rolled back, even though the insert triggered many other events.



ROLLBACK TRIGGER

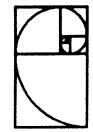
- ROLLBACK TRIGGER rolls back:
  - nested triggers,
  - but not outer transactions.
- Advantage:
  - One can rollback specific inserts, updates or deletes in a transaction rather than the whole transaction.



ROLLBACK TRIGGER

- Example:

```
CREATE TRIGGER trig1
ON Table1
FOR INSERT
AS
    INSERT Table2
    VALUES(5)

CREATE TRIGGER trig2
ON Table2
FOR INSERT
AS
    INSERT Table3
    VALUES(6)
    IF (@@error = 1205)
            ROLLBACK TRAN
```

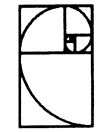


ROLLBACK TRIGGER

- Consider the following transaction:

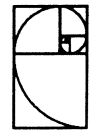
```
BEGIN TRAN
    INSERT Table4
    VALUES(9)
    INSERT Table5
    VALUES(10)
    INSERT Table1
    VALUES(20)
COMMIT TRAN
```

- Let's say the error 1205 is detected when trigger trig2 does the insert into Table3.
- The above set of SQL and triggers will cause the entire transaction to get rolled back.



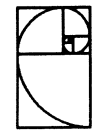
ROLLBACK TRIGGER

- This may not be the desired behavior especially for long running transactions.
- If we replaced the ROLLBACK TRAN in trigger trig2 with ROLLBACK TRIGGER then:
  - The 1205 error will only cause the two inserts in trig1 and trig2 and the insert into Table1 in the transaction to be rolled back.
  - The inserts to tables Table4 and Table5 will still be committed by the transaction.
  - This feature therefore gives application the choice between rolling back the entire transaction or one specific DML statement.



Extended Error Data

- Lets the client find out which columns involved in an error.
- Available via CT-lib only.
- Application can use column name to determine fields in the table that are at fault.

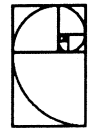


Extended Error Data

- Mocked-up isql-style example:

```
insert t_logins values ("bob", "Bob Epstein")
Msg 2601, Level 14, State 3:
Line 1:
Attempt to insert duplicate key row in object
't_logins' with unique index 'i_login'.

index       database    object      column
-------     ----------  -------     ---------
i_login     emps        t_logins    login
```

- Application could highlight field(s) with names returned in the "column" columns. The user can then correct the problem and redo the DML statement.



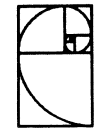
Extended Error Data

- Extended Error Data are returned for:

•	Msg 233 and 515:	Column may not be null
•	Msg 546:	Foreign key constraint violation
•	Msg 547:	Dependent foreign key constraint violation
•	Msg 548:	Check constraint violation
•	Msg 552:	Rule violation
•	Msg 2601:	Duplicate key

- These errors are typical of data-entry mistakes.
- Different error messages return different information. For example:

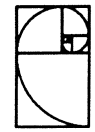
  Rule violations return Rule Name.
  Duplicate Key violations return Index Name.



Extended Error Data

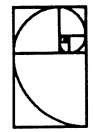
- Extended Error Data can be sent with a RAISERROR.
- Mocked-up isql-style example:

```
raiserror 20100 "Login must be at least 5 characters
long", "column" = "login", "transtate" =
@@transtate

Msg 20100, Level 16, State 1:
Server 'personnel', Line 56:
Login must be at least 5 characters long

column      transtate
---------   ----------
login       3
```



Extended Error Data

- In earlier releases, the programmer could return information via a raiserror message so that the application parsed the message and did the appropriate act, such as highlighting the column "login" in the above example.
- In System 10 SQL Server, the programmer can design a raiserror message for the end-user, and put the information needed by the application in the Extended Error Data. This eliminates the need for parsing the error message.

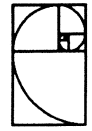


Trigger Self-Recursion

- If an insert trigger on table A does an insert to table A, does the insert performed by the trigger fire the trigger again?

  insert table A values (...)
  This fires an insert trigger, which does:
  insert table A values (...)
  ===> which fires an insert trigger which does:
  insert table A values(...)
  ===> which fires an insert trigger which does:
  insert table A values (...)
  ....

- Pre System 10, this behavior was prevented by not firing the trigger for the second insert into table A from an insert trigger for table A.

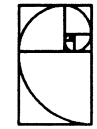


Trigger Self-Recursion

- System 10 SQL Server gives you an option to have the triggers fire for the second and subsequent inserts in the example shown above.
- This behavior of trigger firing itself again is called Self-Recursion.
- The following SET option can be used to turn on this behavior:

  SET SELF_RECURSION ON.

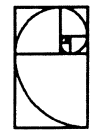
- System 10 SQL Server defaults to self-recursion off.



Budget Rollup Example

- Problem: [Transitive Closure] The budget of one of the departments is increased, which should trigger a corresponding increase in the budget of the parent department and then its parent department etc.
- Schema of table "t_budget"

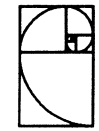
Unit	Parent_unit	Budget
my_dept	my_division	10
my_division	company_wide	100
company_wide	NULL	1000



Budget Rollup Example

- Update trigger on t_budget:

```
SET SELF_RECURSION ON
IF EXISTS(SELECT * FROM inserted
        WHERE parent_unit IS NOT NULL)
BEGIN
    SET SELF_RECURSION ON
    UPDATE t_budget
    SET t_budget.budget = t_budget.budget +
        inserted.budget - deleted.budget
    FROM inserted, deleted
    WHERE t_budget.unit = inserted.parent_unit
        AND t_budget.unit = deleted.parent_unit
END
```

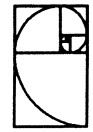


Budget Rollup Example

- The following DML statement updates the budget of my_depts:

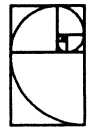
```
UPDATE t_budget
SET budget = budget + 3
WHERE unit = "my_dept"
```

- "my_dept" budget is increased from10 to 13 and the first instantiation of the trigger is fired.



Budget Rollup Example

- This trigger updates the budget of "my_division". The second instantiation of the trigger is now fired.
- This updates the budget of "company_wide". This triggers the third instantiation.
- But that trigger doesn't do an update because "company_wide" has no parent. Since no rows are affected, no new instantiation of the trigger is fired. The self-recursion halts at this point.

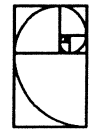


Budget Rollup Example

- Table t_budget after the update:

Unit	Parent_unit	Budget
my_dept	my_division	13
my_division	company_wide	103
company_wide	NULL	1003

- This series of updates would have been very tricky to write without self recursion.
- Also a general solution which supported an arbitrary hierarchy of departments with no code change in triggers would have been almost impossible.



Row Identity

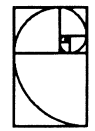
- Most applications need unique sequential identifiers for their row data. Examples are:

  Invoice numbers.
  Employee numbers.
  Account numbers.

- Current Solution is to write sequential number generators using stored procedures or triggers.

- Some of the problems with this approach are:

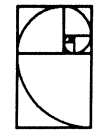
  - Performance degradation.
  - Repeated overhead in application code to ensure accuracy and uniqueness of the numbers generated.



Row Identity

- One such approach used by applications in pre-System 10 SQL Server is the following piece of SQL:

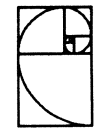
```
BEGIN TRANSACTION
    DECLARE @Maxid INT

    SELECT @Maxid = 1 + MAX(EMP_ID)
    FROM employee HOLDLOCK

    INSERT INTO employee
    VALUES (@Maxid,....)

COMMIT TRANSATION
```



Row Identity

- System 10 SQL Server will support a new column property called "IDENTITY".
- This will simplify generation of primary keys and unique identifiers enormously.
- A column can be designated to be the "IDENTITY" column at table creation time:

```
CREATE TABLE employees
    (emp_id NUMERIC(8, 0) IDENTITY,
     emp_name CHAR(25) NOT NULL,
     emp_addr CHAR(50) NOT NULL)
```



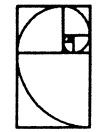
Row Identity - Inserting values

- Inserting rows in this table is simple. The value for column marked as IDENTITY should not be specified. The other values should be specified as usual.

```
INSERT employees
VALUES ("Madeleine B.", "Emeryville, CA")
INSERT employees
VALUES ("Ann Knepper", "Emeryville, CA")
```

- A select from employees will show the following values:

emp_id	emp_name	emp_addr
1	Madeleine B.	Emeryville, CA
2	Ann Knepper	Emeryville, CA

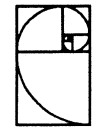


Row Identity - Adding to existing tables

- ALTER TABLE command has been enhanced to support adding IDENTITY column to existing tables.
- This ALTER TABLE command will also assign unique values to existing rows in the table.
- Example:

```
ALTER TABLE existing_table
        ADD id NUMERIC(8, 0) IDENTITY
```

This will also assign values into the id column for existing rows starting from 1.

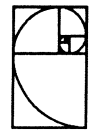


Row Identity - Pseudo Column

- A new way of referring to the IDENTITY column is available in System 10 SQL Server. You can issue the following SQL:

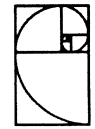
```
SELECT SYB_IDENTITY
FROM employees
```

- The above reference to SYB_IDENTITY will translate into a reference to the column with property IDENTITY in the table, so in the above example it is the *emp_id* column.



Row Identity - Global Variable

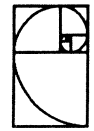
- The value generated by the server for the IDENTITY column is stored in a new global variable called **@@identity**.
- This value is available to stored procedures and triggers. An INSERT trigger can use @@identity to get the value just inserted.



Row Identity - Explicit Inserts

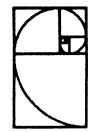
- Occasionally the DBA may need to manually insert values into the IDENTITY column directly.
- One case this may be needed is if the original size of column used to hold IDENTITY values is not big enough, so the table may have to be restructured.
- A new SET command will allow inserts to the IDENTITY column:

```
SET IDENTITY_INSERT <table name> ON
e.g
SET IDENTITY_INSERT employees ON
INSERT employees(emp_id, emp_name, emp_addr)
VALUES(4, "Laurie J.", "Emeryville, CA")
```



Row Identity - Other commands

- Several other commands have been enhanced to be used with IDENTITY columns.
- SELECT/INTO: A target IDENTITY column can now be specified.
- Bulk Copy: Identity columns can be copied in/out. Both implicit and explicit inserts into the column are supported.
- **sp_help**: The IDENTITY columns will be identified.
- **sp_addtype**: Datatypes with IDENTITY property can be added.



Agenda

- Expectations
- Architectural overview
- Public files
- Initializing an Open Server
- API Paradigm Changes
- Command and result processing
- Major new features
- Tracing

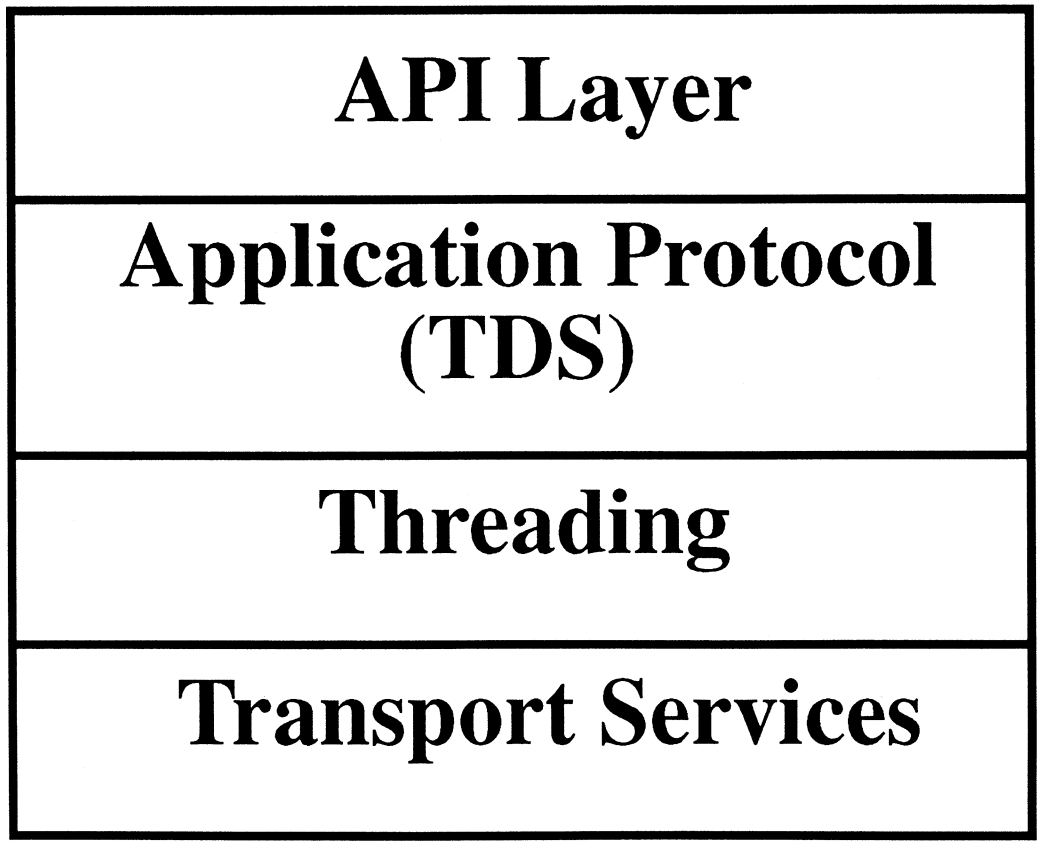
By Dan Sifter

- Migration
- Documentation
- Sample programs

Expectations

- Assume familiarity with 2.0
- Architectural overview
- Initialize a System 10 Open Server
- Process commands and results in a System 10 Open Server
- Familiar with new System 10 features
- Familiar with tracing facilities
- Migrate to System 10

Architectural Overview



Transport Services

- Common transport interface (Transport Control Library)
- Multiple Listener support
- Generic protocol driver interface
  - 3rd party drivers
- Dynamic driver loading on most platforms
  - Configuration file to install new drivers

Client/Server Library

- Data type conversion routines
- Arithmetic operations
- Localization
  - Character set conversion
  - Sort orders
  - Localized error messages
- Date/Time operations
- Bulk copy
  - Both Open Server and CT-Library

Public Files

- cstypes.h
  - C/S typedefs
  - Data type defines
- cspublic.h
  - Shared C/S defines
  - C/S library prototypes
- ospublic.h
  - Open Server specific defines and typedefs
  - Function prototypes
  - Includes cspublic.h
  - Replaces srv.h
- oserror.h
  - Open Server error defines

- Replaces srverror.h

- oscompat.h
  - Backward compatibility defines
  - Included by ospublic.h
  - Obsolete function prototypes
  - All definitions removed in next major release

- Localization files
  - Character sets
  - Localized error messages
  - Localization name mappings

- Libraries
  - libblk.a - Bulk copy (public)
  - libcomn.a - Common routines (private)
  - libcs.a - C/S Library (public)
  - libct.a - CT-Library (public)

- libintl.a - Internationalization routines (private)
- libtcl.a - Transport Control Library (private)
- libsrv.a - Open Server (public)
- libsybdb.a - DB-Library (public)
- Net-Library drivers (many)

Initialization

- Context allocation
  - Allocated in C/S Library (cs_ctx_alloc)
  - Default localization information
  - Global state information
- Setting version
  - Set Open Server library version
  - Context information
  - Must be FIRST call to Open Server library
  - srv_version
- Configuration
  - Optional
  - Properties

- Resource initialization
  - Allocate memory and other resources
  - Set server name (or use DSLISTEN)
  - srv_init
- Start Open Server
  - Start listener and accepting connections
  - srv_run
  - Doesn't return until SRV_STOP event

Properties

- Server and thread level properties
- Server properties use to configure server wide characteristics
- Thread properties used to configure thread characteristics and determine thread information
- Replace srv_config, srv_sfield, and srv_pfield

```
#define MAXLOGSIZE      128000
#define FPRINTF         (CS_VOID)fprintf

/*
** CONFIGLOG
**
**      This routine configures the Open Server log file to be
**      truncated on start-up and if it reaches a maximum size
*/
CS_RETCODE      configlog(cp)
CS_CONTEXT      *cp;
{
        CS_BOOL         truncate;
        CS_INT          maxsize;

        /* Initialization                                                */
        truncate = CS_TRUE;
        maxsize = MAXLOGSIZE;
```

```
        /*
        ** Configure Open Server to truncate the log file on start-up
        */
        if(srv_props(cp, CS_SET, SRV_S_TRUNCATELOG, &truncate,
                sizeof(truncate), (CS_INT *)NULL) == CS_FAIL)
        {
                FPRINTF(stderr, "srv_props SRV_S_TRUNCATELOG failed\n");
                return(CS_FAIL);
        }

        /*
        ** Configure Open Server to truncate the log file if it
        ** exceeds a maximum size
        */
        if(srv_props(cp, CS_SET, SRV_S_LOGSIZE, &maxsize,
                sizeof(maxsize), (CS_INT *)NULL) == CS_FAIL)
        {
                FPRINTF(stderr, "srv_props SRV_S_LOGSIZE failed\n");
                return(CS_FAIL);
        }
        return(CS_SUCCEED);
}
```

```
/*
** CHECKSTACK
**
**      This routine checks the remaining stacksize left for the
**      specified thread.
*/
CS_RETCODE      checkstack(spp)
SRV_PROC        *spp;
{
        CS_INT  stackleft;

        if(srv_thread_props(spp, CS_GET, SRV_T_STACKLEFT, &stackleft,
                sizeof(stackleft), (CS_INT *)NULL) == CS_FAIL)
        {
                FPRINTF(stderr, "srv_thread_props SRV_T_STACKLEFT failed\n");
                return(CS_FAIL);
        }

        /*
        ** Print out remaining stack size for specified thread
        */
        FPRINTF(stderr, "stackleft(0x%d) for thread(0x%x)\n", stackleft, spp);
        return(CS_SUCCEED);
}
```

API Paradigm Changes

- Public data structures shared with CT-Lib
- Describe/Bind/Transfer (CS_DATAFMT)
- Automatic data type conversion
- Internal data structure abstraction
  - No internal data structure definitions will be shipped
  - Binary compatibility
- State checking

```
/*
** define the data format structure used by OpenClient
*/
typedef struct _cs_datafmt
{
    /*
    ** These members are commonly used.
    */
    CS_CHAR     name[CS_MAX_NAME];      /* the name of the data */
    CS_INT      namelen;        /* the actual length of the name */

    /*
    ** The datatype field indicates what OpenClient or user defined
    ** datatype is being represented.
    **
    ** The format field tells whether or not data should be padded
    ** to the full length of the variable. This will only be used
    ** if the type of the variable is character or binary. The
    ** format field also tells whether the data should be
    ** null-terminated (for char variables, only).
    **
    ** See the types man page for more detail on this
    */
    CS_INT      datatype;       /* the datatype of the data */
    CS_INT      format;         /* format of the data (i.e. blank padded) */

    CS_INT      maxlength;      /* the max length the data might be */
```

```
    CS_INT      scale;          /* used if datatype needs it (i.e. numeric) */
    CS_INT      precision;      /* used if datatype needs it (i.e. numeric) */
    CS_INT      status;         /* data status, e.g.,  key, return param */

    /*
    ** If binding data, the count field tells how many rows should be bound
    */
    CS_INT      count;

    /*
    ** These elements are not usually used. They are here to support
    ** user-defined datatypes and localization (internalization) info.
    */
    CS_INT      usertype;       /* user-defined datatype code */
    CS_LOCALE   *locale;
} CS_DATAFMT;
```

CS_DATAFMT

- Data description structure
- Defined in cstypes.h
- Structure definitions
  - name/namlen
    - Column/parameter names
  - datatype
    - Type of parameter or column data type (CS_INT_TYPE, CS_CHAR_TYPE, etc.)
  - format
    - Format of character/binary data. Used in binding.
  - maxlength
    - Maximum length of data type (4 bytes for CS_INT_TYPE, etc.)
  - scale/precision
    - Numeric/decimal only
  - status
    - Status information on column or parameter (i.e. key or return parameter, etc.)

- count
  - Array binding in CT-Lib. Not used in Open Server. Must be 0 or 1.
- usertype
  - User defined data type
- locale
  - Localization information associated with this data item.

Data Description

- Parameter and column description
- srv_descfmt
- Command argument (CS_GET, CS_SET)
- Type argument
  - SRV_RPCDATA
  - SRV_ROWDATA
  - SRV_CURDATA
  - SRV_UPCOLDATA
  - SRV_KEYDATA
  - SRV_ERRORDATA
  - SRV_DYNDATA
  - SRV_NEGDATA
  - SRV_MSGDATA

- Item argument
  - Column or parameter number being described
- CS_DATAFMT argument
  - Contains description of data from client (CS_GET)
  - Contains description of data to send to client (CS_SET)

Binding Program Variables

- Source or destination program variables
- srv_bind
- Command argument
  - CS_GET, read data into program variable
  - CS_SET, read data out of program variable
- Type argument
  - Same as srv_descfmt
- Item argument
  - Column or parameter to which program variable is being bound

- CS_DATAFMT argument
  - Description of program variable
  - Automatic conversion done
- varaddrp argument
  - Pointer to location of program variable
- varlenp argument
  - Pointer to location of program variable containing actual length of data
- indp argument
  - Pointer to location of program variable containing null indicator
  - Actual length of 0 no longer assumed to be null
- Can change contents of pointer arguments without rebinding

Transferring Data

- Transfer data from/to program variables
- srv_xferdata
- Command argument
  - CS_GET, transfer client data into program variables
  - CS_SET, transfer Open Server application data to client
- Type argument
  - Same as srv_descfmt/srv_bind
- Program variable pointer arguments to srv_bind must be valid until after srv_xferdata is called

```
#define COL1NAME        "CS_INT_TYPE"
#define COL2NAME        "CS_MONEY_TYPE"
#define NUMROWS         10

/*
** LANGHANDLER
**
**      Language handler that returns NUMROWS rows.  Each row
**      contains 2 columns one of type CS_INT_TYPE, and one
**      of type CS_MONEY_TYPE.
*/
CS_RETCODE      langhandler(spp)
SRV_PROC        *spp;
{

        CS_DATAFMT      col1fmt;
        CS_INT          col1data;
        CS_INT          col1len;
        CS_SMALLINT     col1ind;
        CS_DATAFMT      col2fmt;
        CS_INT          col2data;
        CS_INT          col2len;
        CS_SMALLINT     col2ind;
        int             i;
```

```
/*
** Initialize and describe column 1
*/
srv_bzero(&col1fmt, sizeof(col1fmt));
srv_bmove(COL1NAME, col1fmt.name, strlen(COL1NAME));
col1fmt.namelen = strlen(COL1NAME);
col1fmt.datatype =  CS_INT_TYPE;
col1fmt.format =  CS_FMT_UNUSED;
col1fmt.maxlength = sizeof(CS_INT);

if(srv_descfmt(spp, CS_SET, SRV_ROWDATA, 1, &col1fmt) == CS_FAIL)
{
        FPRINTF(stderr, "srv_descfmt col1 failed\n");
        (CS_VOID)srv_senddone(spp, SRV_DONE_FINAL|SRV_DONE_ERROR,
                CS_TRAN_UNDEFINED, (CS_INT)0);
        return(CS_SUCCEED);
}
```

```
/*
** Initialize and describe column 2
*/
srv_bzero(&col2fmt, sizeof(col2fmt));
srv_bmove(COL2NAME, col2fmt.name, strlen(COL2NAME));
col2fmt.namelen = strlen(COL2NAME);
col2fmt.datatype =  CS_MONEY_TYPE;
col2fmt.format =  CS_FMT_UNUSED;
col2fmt.maxlength = sizeof(CS_MONEY);

if(srv_descfmt(spp, CS_SET, SRV_ROWDATA, 2, &col2fmt) == CS_FAIL)
{
        FPRINTF(stderr, "srv_descfmt col2 failed\n");
        (CS_VOID)srv_senddone(spp, SRV_DONE_FINAL|SRV_DONE_ERROR,
                CS_TRAN_UNDEFINED, (CS_INT)0);
        return(CS_SUCCEED);
}
```

```
/*
** Bind the program variable for column 1.  We just reuse the
** original col1fmt because the program variable is of the
** same type as the client data type.
**
** NOTE:  We don't have to initialize the col1data, col1len, and
** col1ind variables yet because this data only has to be valid
** at srv_xferdata time.
*/
if(srv_bind(spp, CS_SET, SRV_ROWDATA, 1, &col1fmt, &col1data,
        &col1len, &col1ind) == CS_FAIL)
{
        FPRINTF(stderr, "srv_bind col1 failed\n");
        (CS_VOID)srv_senddone(spp, SRV_DONE_FINAL|SRV_DONE_ERROR,
                CS_TRAN_UNDEFINED, (CS_INT)0);
        return(CS_SUCCEED);
}
```

```
/*
** Bind the program variable for column 2.  We change the
** original col2fmt structure to indicate that the program
** variable is a CS_INT_TYPE, not a CS_MONEY_TYPE.  Open
** Server will perform the data conversion automatically
** at srv_xferdata time.
**
** NOTE:  We don't have to initialize the col2data, col2len, and
** col2ind variables yet because this data only has to be valid
** at srv_xferdata time.
*/
col2fmt.datatype = CS_INT_TYPE;
col2fmt.maxlength = sizeof(CS_INT);

if(srv_bind(spp, CS_SET, SRV_ROWDATA, 2, &col2fmt, &col2data,
        &col2len, &col2ind) == CS_FAIL)
{
        FPRINTF(stderr, "srv_bind col2 failed\n");
        (CS_VOID)srv_senddone(spp, SRV_DONE_FINAL|SRV_DONE_ERROR,
                CS_TRAN_UNDEFINED, (CS_INT)0);
        return(CS_SUCCEED);
}
```

```
/*
** Initialize the NULL indicators to indicate that we are always
** sending valid data
*/
col1ind = CS_GOODDATA;
col2ind = CS_GOODDATA;

/*
** Initialize the length of the program variables for the
** transfer
*/
col1len = sizeof(CS_INT);
col2len = sizeof(CS_INT);
```

```
/*
** Now transfer NUMROWS of data to the client.  We use the
** loop variable as the data to return to the client.
*/
for(i = 0; i < NUMROWS; i++)
{
        col1data = i;
        col2data = i;

        if(srv_xferdata(spp, CS_SET, SRV_ROWDATA) == CS_FAIL)
        {
                FPRINTF(stderr, "srv_xferdata row(%d) failed\n", i);
                (CS_VOID)srv_senddone(spp,
                        SRV_DONE_FINAL|SRV_DONE_ERROR,
                        CS_TRAN_UNDEFINED, 0);
                return(CS_SUCCEED);
        }
}
```

```
        /*
        ** Send the final done to the client
        */
        if(srv_senddone(spp, SRV_DONE_COUNT | SRV_DONE_FINAL,
                CS_TRAN_UNDEFINED, i) == CS_FAIL)
        {
                FPRINTF(stderr, "srv_senddone FINAL failed\n");
        }
        return(CS_SUCCEED);
}
```

New Data Types

- Text/Image
- Numeric/Decimal
- Long character/binary
  - CT-Lib and Open Server only
  - 4 byte length character or binary data stream
- Sensitivity/Boundary security data types

New Events

- Bulk I/O
  - writetext and bulk copy
- Cursor
- Dynamic
- Message
- Options

Internationalization

- Localization files for error messages
- Error messages reported in either client or server national language
- Error message language configured on a thread basis
- Character set translations
- Support for client option commands to change language or character set

Cursors/Dynamic SQL

- API's to support declaration of cursors and dynamically prepared SQL
- Receive/respond to cursor commands
- Support scrollable cursors
- Receive/respond to dynamic SQL commands
- Declare cursor on a dynamically prepared statement
- Run ESQL generated programs through Open Server
- srv_cursor_props
- srv_dynamic

Supported Cursor Commands

- All commands received in cursor event handler
- Declare
- Open
- Fetch
  - Settable fetch count
- Update/Delete
  - Relative and absolute
- Close (Dealloc)

Negotiated Logins

- Generic mechanism to perform client/server login negotiation
- Support for both direct client and server-to-server connections
- Performed in connect handler
- Allow applications to implement secure login schemes in Open Server
- srv_negotiate

Messages

- Generic one-way communication mechanism
- Application defined message IDs
- Parameter support
- Parameters can haven any data type except text/image
- srv_msg

```
#define FPRINTF          (CS_VOID)fprintf
#define APPMSG1          (SRV_MAXRESMSG + 1)
#define APPMSG2          (SRV_MAXRESMSG + 2)
#define PARAMSIZE        255
#define MAXPARAMS        10

/*
** MSGHANDLER
**
**      Message handler that reads a message from a client.  Any
**      parameters associated with the message are also read and
**      converted to character data
*/
CS_RETCODE       msghandler(spp)
SRV_PROC         *spp;
{
        CS_INT           msgid;
        CS_INT           msgstatus;
        CS_INT           numparams;
        CS_INT           size;
        CS_DATAFMT       fmt[MAXPARAMS];
        CS_DATAFMT       *fmtp;
        CS_CHAR          buf[MAXPARAMS * PARAMSIZE];
        CS_CHAR          *bp;
        CS_INT           ind[MAXPARAMS];
        CS_INT           *indp;
        int              i;
```

```
        /* Initialization                                                    */
        fmtp = fmt;
        bp = buf;
        indp = ind;
        srv_bzero(fmtp, sizeof(fmt));

        /*
        ** Read the message from the client
        */
        if(srv_msg(spp, CS_GET, &msgid, &msgstatus) == CS_FAIL)
        {
                FPRINTF(stderr, "srv_msg failed\n");
                (CS_VOID)srv_senddone(spp, SRV_DONE_FINAL|SRV_DONE_ERROR,
                        CS_TRAN_UNDEFINED, (CS_INT)0);
                return(CS_SUCCEED);
        }
```

```
/*
** Based on the message ID the Open Server application would
** take some interesting action
*/
switch(msgid)
{
case APPMSG1:
        processappmsg1(spp);
        break;

case APPMSG2:
        processappmsg2(spp);
        break;

default:
        /*
        ** Unknown message ID
        */
        FPRINTF(stderr, "unknown msgid \n");
        (CS_VOID)srv_senddone(spp, SRV_DONE_FINAL|SRV_DONE_ERROR,
                CS_TRAN_UNDEFINED, (CS_INT)0);
        return(CS_SUCCEED);
}
```

```
/*
** Check to see if this message has any parameters.  If there
** aren't any parameters we are all done
*/
if(!(msgstatus & SRV_HASPARAMS))
{
        FPRINTF(stderr, "Message has no parameters\n");
        (CS_VOID)srv_senddone(spp, SRV_DONE_FINAL, CS_TRAN_UNDEFINED,
                (CS_INT)0);
        return(CS_SUCCEED);
}

/*
** Determine the number of parameters in the message
*/
if(srv_numparams(spp, &numparams) == CS_FAIL)
{
        FPRINTF(stderr, "srv_numparams failed\n");
        (CS_VOID)srv_senddone(spp, SRV_DONE_ERROR | SRV_DONE_FINAL,
                CS_TRAN_UNDEFINED, (CS_INT)0);
        return(CS_SUCCEED);
}
```

```
/*
** Make sure we have room for all of the parameters
*/
if(numparams > MAXPARAMS)
{
        FPRINTF(stderr, "Too many parameters\n");
        (CS_VOID)srv_senddone(spp, SRV_DONE_ERROR | SRV_DONE_FINAL,
                CS_TRAN_UNDEFINED, (CS_INT)0);
        return(CS_SUCCEED);
}

/*
** We could use srv_descfmt at this point to get a description
** of the parameters being sent by the client.  However, since
** Open Server does automatic data type conversion we'll just
** bind character arrays.  All of the parameters will be
** automatically converted to characters.
*/
```

```
/*
** Bind program variables for each of the parameters
*/
for(i = 0; i < numparams; i++)
{
        /*
        ** Initialize the CS_DATAFMT structure for this
        ** program variable.  We want all parameters
        ** converted to chacters.  The size of the
        ** program buffer is put into the maxlength field.
        ** We also want all of the program buffers to be
        ** null terminated.
        */
        fmtp->datatype = CS_CHAR_TYPE;
        fmtp->maxlength = PARAMSIZE;
        fmtp->format = CS_FMT_NULLTERM;

        if(srv_bind(spp, CS_GET, SRV_MSGDATA, (i + 1), fmtp, bp,
                (CS_INT *)NULL, indp) == CS_FAIL)
        {
                FPRINTF(stderr, "srv_bind failed\n");
                (CS_VOID)srv_senddone(spp,
                        SRV_DONE_ERROR | SRV_DONE_FINAL,
                        CS_TRAN_UNDEFINED, (CS_INT)0);
                return(CS_SUCCEED);
        }
```

```
                /*
                ** Bump buffer pointers for next time around loop
                */
                fmtp++;
                bp++;
                indp++;
        }
        /*
        ** Now transfer the parameters into the bound program variables
        */
        if(srv_xferdata(spp, CS_GET, SRV_MSGDATA) == CS_FAIL)
        {
                FPRINTF(stderr, "srv_xferdata failed\n", i);
                (CS_VOID)srv_senddone(spp, SRV_DONE_FINAL|SRV_DONE_ERROR,
                        CS_TRAN_UNDEFINED, (CS_INT)0);
                return(CS_SUCCEED);
        }

        /*
        ** Send the final done to the client
        */
        if(srv_senddone(spp, SRV_DONE_FINAL, CS_TRAN_UNDEFINED, (CS_INT)0)
                == CS_FAIL)
        {
                FPRINTF(stderr, "srv_senddone FINAL failed\n");
        }
        return(CS_SUCCEED);
}
```

Bulk I/O

- Support for writetext/readtext/writebulk client commands
- API's to parse bulk copy columns
- Bulk Event Handler
- srv_text_info/srv_get_text

```
#define BULKBUFSIZE     1024
#define FPRINTF         (CS_VOID)fprintf

/*
**  BULK_HANDLER
**
**      Open Server bulk event handler to process "writetext <columnname>
**      txtptr data" commands from a client application
*/
CS_RETCODE      bulkhandler(spp)
SRV_PROC        *spp;
{
        CS_INT          bulktype;
        CS_IODESC       iodesc;
        CS_BYTE         buf[BULKBUFSIZE];
        CS_INT          outlen;
        CS_BOOL         done;
        CS_RETCODE      ret;

        /* Initialization                                                */
        done = CS_FALSE;
```

```
/*
** Get the current bulk data type being sent by the client.
** The Open Server bulk handler is called for both bulk
** text/image data and bulk copy data
*/
if(srv_thread_props(spp, CS_GET, SRV_T_BULKTYPE, &bulktype,
        sizeof(bulktype), (CS_INT)NULL) == CS_FAIL)
{
        FPRINTF(stderr, "srv_thread_props SRV_T_BULKTYPE failed\n");
        (CS_VOID)srv_senddone(spp, SRV_DONE_FINAL|SRV_DONE_ERROR,
                CS_TRAN_UNDEFINED, (CS_INT)0);
        return(CS_SUCCEED);
}

/*
** Make sure the client is sending either text or image.  This
** bulk handler does not support bulk copy
*/
if(bulktype == SRV_BULKLOAD)
{
        FPRINTF(stderr, "Unsupported bulk data type\n");
        (CS_VOID)srv_senddone(spp, SRV_DONE_FINAL|SRV_DONE_ERROR,
                CS_TRAN_UNDEFINED, (CS_INT)0);
        return(CS_SUCCEED);
}
```

```
/*
** Get the total length of the text/image data being sent by the
** client.   The total length of the text/image data is returned
** in the total_txtlen field of the CS_IODESC.
**
** Since a client can only send rows with one column
** of text/image data, we just hard code the item number to 1
*/
if(srv_text_info(spp, CS_GET, 1, &iodesc) == CS_FAIL)
{
        FPRINTF(stderr, "srv_text_info failed\n");
        (CS_VOID)srv_senddone(spp, SRV_DONE_FINAL|SRV_DONE_ERROR,
                CS_TRAN_UNDEFINED, (CS_INT)0);
        return(CS_SUCCEED);
}
```

```
/*
** The total text/image length can be used to allocate a buffer.
** In this example, we just use a fixed length buffer on the
** stack to read the data in chunks
*/
while(!done)
{
        /*
        ** Read a chunk of data from the client
        */
        if((ret = srv_get_text(spp, buf, sizeof(buf), &outlen))
                == CS_FAIL)
        {
                FPRINTF(stderr, "srv_get_text failed\n");
                (CS_VOID)srv_senddone(spp,
                        SRV_DONE_FINAL|SRV_DONE_ERROR,
                        CS_TRAN_UNDEFINED, (CS_INT)0);
                return(CS_SUCCEED);
        }

        /*
        ** Do something interesting with the text/image data
        ** received from the client here
        */
```

```
                /*
                ** Check the return code from srv_get_text.  CS_END_DATA
                ** is returned when all of the data has been read from
                ** the client
                */
                if(ret == CS_END_DATA)
                {
                        done = CS_TRUE;
                }
        }
        /*
        ** Send the final done to the client
        */
        if(srv_senddone(spp, SRV_DONE_FINAL, CS_TRAN_UNDEFINED, (CS_INT)0)
                == CS_FAIL)
        {
                FPRINTF(stderr, "srv_senddone failed\n");
        }
        return(CS_SUCCEED);
}
```

Capabilities

- Negotiate connection specific command, result, and data types
- Requires a System 10 client to negotiate capabilities
- Default capabilities for non-System 10 clients
- Server rules
- srv_capability_info

```
#define FPRINTF          (CS_VOID)fprintf

/*
** CHECKCAPS
**
**      This routine determines if a client supports the long character and
**      binary data types in response result sets
*/
CS_RETCODE      checkcaps(spp)
SRV_PROC        *spp;
{
        CS_BOOL         nolchar;
        CS_BOOL         nolbin;

        /*
        ** Check to see if this client supports long char data types
        */
        if(srv_capability_info(spp, CS_GET, CS_CAP_RESPONSE, CS_DATA_NOLCHAR,
                &nolchar) == CS_FAIL)
        {
                FPRINTF(stderr, "CS_DATA_NOLCHAR failed\n");
                return(CS_FAIL);
        }
```

```
        /*
        ** Check to see if this client supports long binary data types
        */
        if(srv_capability_info(spp, CS_GET, CS_CAP_RESPONSE, CS_DATA_NOLBIN,
                &nolbin) == CS_FAIL)
        {
                FPRINTF(stderr, "CS_DATA_NOLBIN failed\n");
                return(CS_FAIL);
        }

        /*
        ** Display the results of the client's capabilities
        */
        FPRINTF(stderr, "Long character %s supported\n",
                nolchar ? "is not" : "is");
        FPRINTF(stderr, "Long binary %s supported\n",
                nolbin ? "is not" : "is");
        return(CS_SUCCEED);
}
```

Extended Error Handling

- Allows parameter information to be returned with an informational/error data stream
- Transaction state information can also be returned to client
- srv_sendinfo
- Replaces srv_sendmsg

Other New Features

- Browse mode
- Control rows
- Compute rows
- Orderbys
- Required for a "virtual SQL Server" implementation

Tracing

- Enabled using srv_props(SRV_S_TRACEFLAG)
- SRV_TR_TDSHDR/SRV_TR_TDSDATA
  - TDS header and data
- SRV_TR_MSGQ
  - Message queue tracing (both internal and external)
- SRV_TR_ATTN
  - Attention handling
- SRV_TR_EVENT
  - Event tracing

- SRV_TR_NETREQ/SRV_TR_NETDRIVER/SRV_TR_NETWAKE
  - Transport control library request tracing
  - Net-Library driver tracing
  - Wakeup tracing
- SRV_TR_DEFQUEUE
  - Deferred event queue tracing

Migration

- Requires source code changes to initialization and srv_run return code checking
  - srv_run now returns CS_SUCCEED/CS_FAIL, not SRV_STOP
- Makefile changes to link with additional libraries
- Stub routines
- Problem areas
  - Using undocumented defines/typedefs
  - Accessing fields in internal data structures
    - SRV_PROC
    - SRV_SERVER
    - SRV_CONFIG
    - DBLOGINFO
- Tighter state checking

- srv_senddone argument changes
  - Replaced info argument with transaction state
- Conversion routines
  - Now consistent with SQL Server

```
CS_RETCODE      initserver(cp)
SRV_CONFIG      *cp;
{
        SRV_SERVER      *ssp;           /* Server control structure     */

        /*
        ** Initialize Open Server
        */
        if((ssp = srv_init(cp, (CS_CHAR *)NULL, 0)) == (SRV_SERVER *)NULL)
        {
                (CS_VOID)fprintf(stderr, "%s: srv_init failed\n", servername);
                exit(1);
        }

        /*
        ** Start the Open Server running
        */
        if(srv_run(ssp) != SRV_STOP)
        {
                (CS_VOID)fprintf(stderr, "%s: srv_run failed\n", servername);
                exit(1);
        }
        return(CS_SUCCEED);
}
```

```
#define FPRINTF (CS_VOID)fprintf

CS_RETCODE      initserver()
{
        CS_CONTEXT      *cp;            /* Context Structure            */
        SRV_SERVER      *ssp;           /* Server control structure     */

        /*
        ** Allocate a C/S Library context structure to define the default
        ** localization information.  Global state information will also
        ** be stored by Open Server in this structure during initialization
        */
        if(cs_ctx_alloc(CS_VERSION_100, &cp) != CS_SUCCEED)
        {
                FPRINTF(stderr, "%s: cs_ctx_alloc failed", servername);
                exit(1);
        }
```

```
/*
** Set the Open Server version and context information
*/
if(srv_version(cp, CS_VERSION_100) != CS_SUCCEED)
{
        /*
        ** Release the context structure already allocated
        */
        (CS_VOID)cs_ctx_drop(cp);

        FPRINTF(stderr, "%s: srv_version failed", servername);
        exit(1);
}
```

```
/*
** Initialize Open Server
*/
if((ssp = srv_init((SRV_CONFIG *)NULL, (CS_CHAR *)NULL, 0))
        == (SRV_SERVER *)NULL)
{
        /*
        ** Release the context structure already allocated
        */
        (CS_VOID)cs_ctx_drop(cp);

        FPRINTF(stderr, "%s: srv_init failed\n", servername);
        exit(1);
}
```

```
        /*
        ** Start the Open Server running
        */
        if(srv_run((SRV_SERVER *)NULL) == CS_FAIL)
        {
                /*
                ** Release the context structure already allocated
                */
                (CS_VOID)cs_ctx_drop(cp);

                FPRINTF(stderr, "%s: srv_run failed\n", servername);
                exit(1);
        }
        return(CS_SUCCEED);
}
```

```
#define FPRINTF          (CS_VOID)fprintf

/*
** GETSPID
**
**      This routine prints the spid for the specified thread
**
**      NOTE:  This code would work in releases previous to 10.0.
**      This code will cause a compile time error in System 10.
*/
CS_RETCODE      getspid(spp)
SRV_PROC        *spp;
{
        CS_INT  spid;

        /*
        ** Get the current spid for this thread
        */
        spid = spp->srvp_spid
        FPRINTF(stderr, "spid(0x%d) for thread(0x%x)\n", spid, spp);
        return(CS_SUCCEED);
}
```

```
/*
** GETSPID
**
**      This routine prints the spid for the specified thread
**
*/
CS_RETCODE      getspid(spp)
SRV_PROC        *spp;
{
        CS_INT  spid;

        /*
        ** Get the current spid for this thread
        */
        if(srv_thread_props(spp, CS_GET, SRV_T_SPID, &spid,
                sizeof(spid), (CS_INT *)NULL) == CS_FAIL)
        {
                FPRINTF(stderr, "srv_thread_props SRV_T_SPID failed\n");
                return(CS_FAIL);
        }
        FPRINTF(stderr, "spid(0x%d) for thread(0x%x)\n", spid, spp);
        return(CS_SUCCEED);
}
```

Documentation

- Completely new manual
- Topics pages
- Additional sample programs

Sample Programs

- lang.c
  - Language handler
- fullpass.c
  - Passthru gateway
- regproc.c
  - Registered procedures
- intlchar.c
  - Internationalization
- sigalarm.c
  - UNIX SIGALRM handling

- exfds.c
  - External file descriptor handling on UNIX
- multthrd.c
  - Threading
- ctosdemo.c
  - Full gateway sample program
    - Attention handling
    - Browse mode
    - Bulk copy
    - Compute rows
    - Control rows
    - Cursors
    - Dynamic SQL
    - Language
    - Messages
    - Options
    - Orderbys
    - Remote Procedure Calls
    - Text and image

OmniSQL Gateway 10.0

Product Overview

By Steve Olson

Overview - What is Omni?

- Omni is an Open Server application that attempts to present a common interface and appearance to multiple sources of data

  "A common interface from the desktop to the Enterprise"

- Omni enables applications written for SQL Server to access other SQL and non-SQL Server data sources

- Omni is not a data manager. It can be thought of more properly as a 'schema manager'.

- Engineering design goal is to appear identical to a SQL Server - we're almost there today.

Overview - Background

- Early versions of SQL Gateways enable Open Client applications to interact with non-Sybase databases.

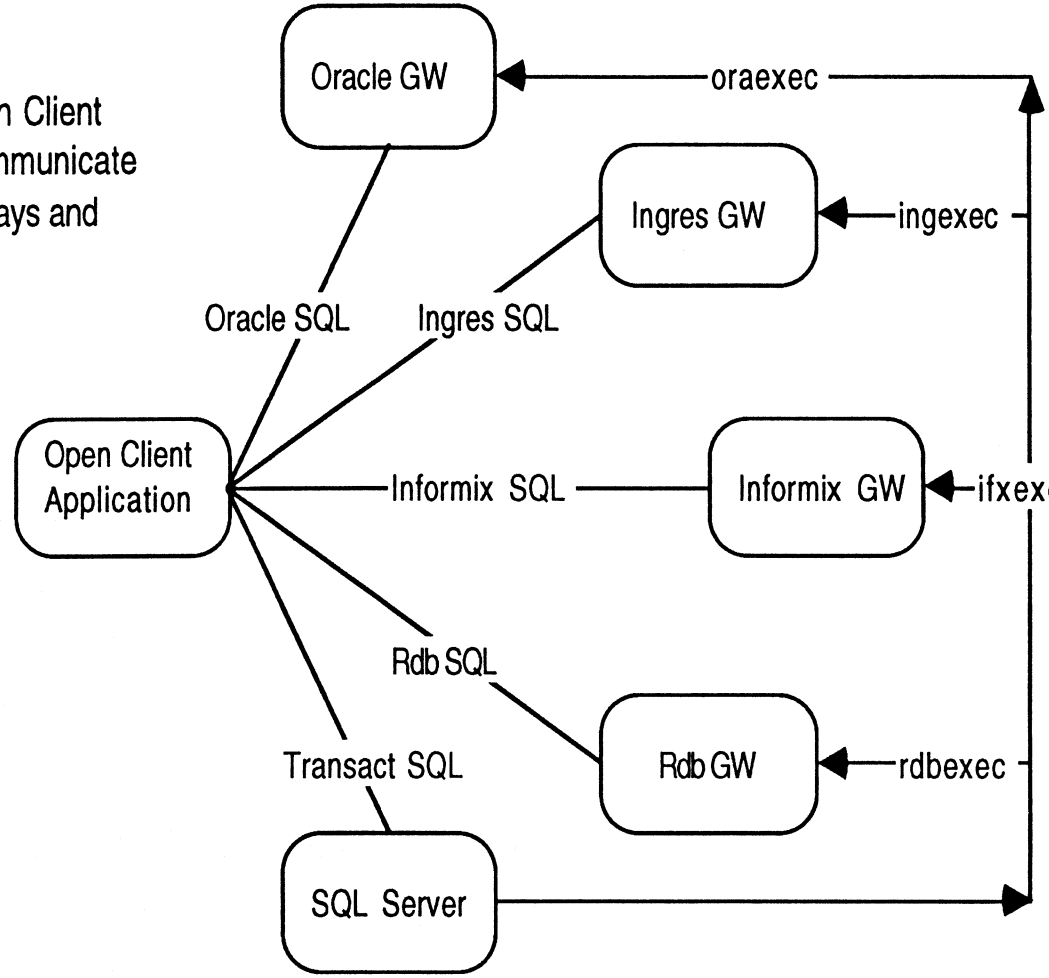
  Oracle
  Ingres
  Rdb
  Informix
  DB2

- Application are required to send native SQL to the gateway; no translation performed

- For each source of data, a separate Open Client connection must be established

Overview - Gateway Architecture

client uses Open Client interface to communicate with SQL Gateways and SQL Server



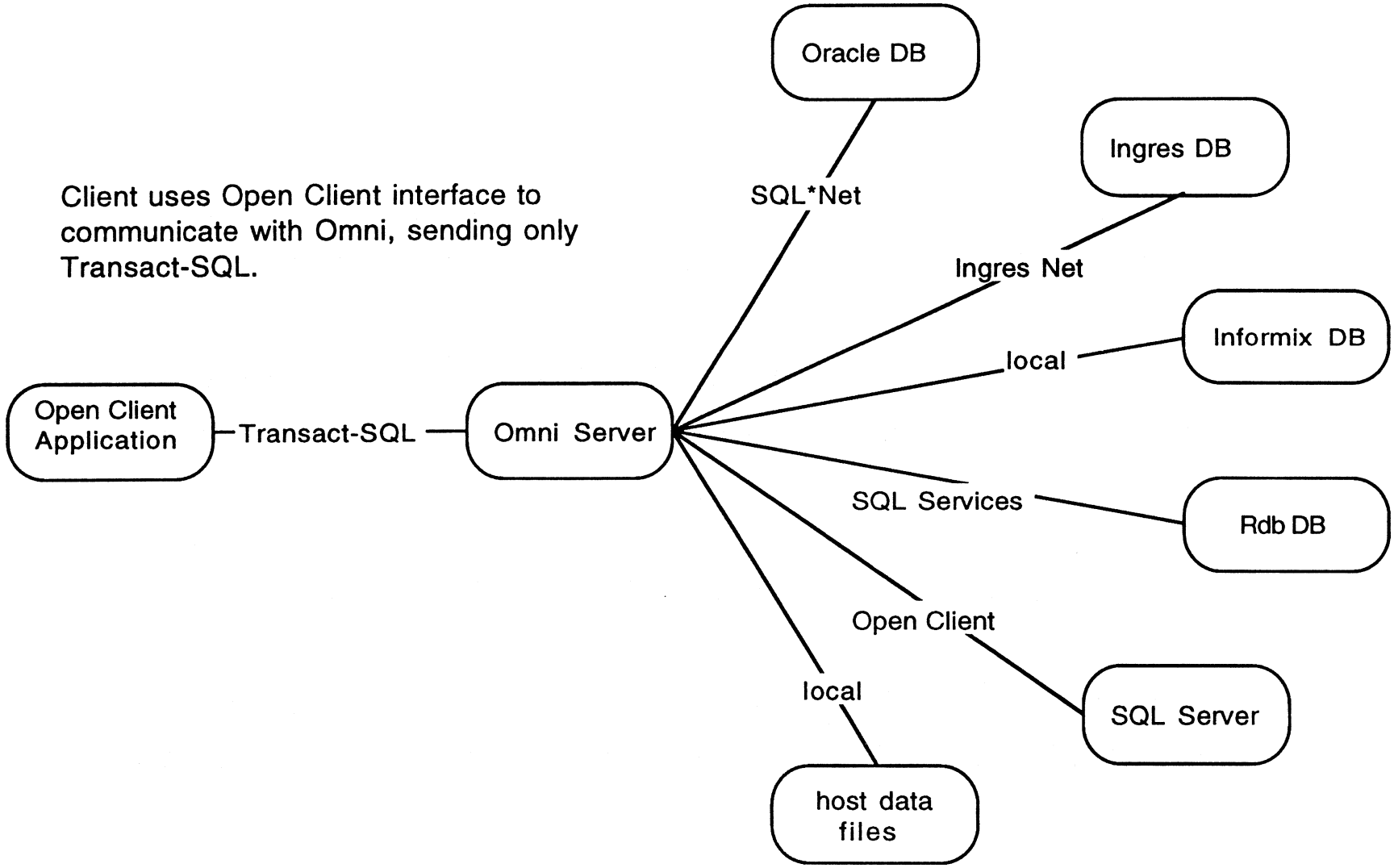
RPC's sent from SQL Server to SQL Gateways

Overview - OmniSQL Gateway

- Addresses the problem of non-uniform flavors of SQL when addressing various RDBMS through SQL gateways
- Single Language - Transact-SQL - only requirement for the application
- Access to remote heterogeneous data sources handled transparently by OmniSQL Gateway.
- To an Open Client application, the OmniSQL Gateway looks and behaves very much like a SQL Server.

Overview - Omni Architecture

Client uses Open Client interface to communicate with Omni, sending only Transact-SQL.

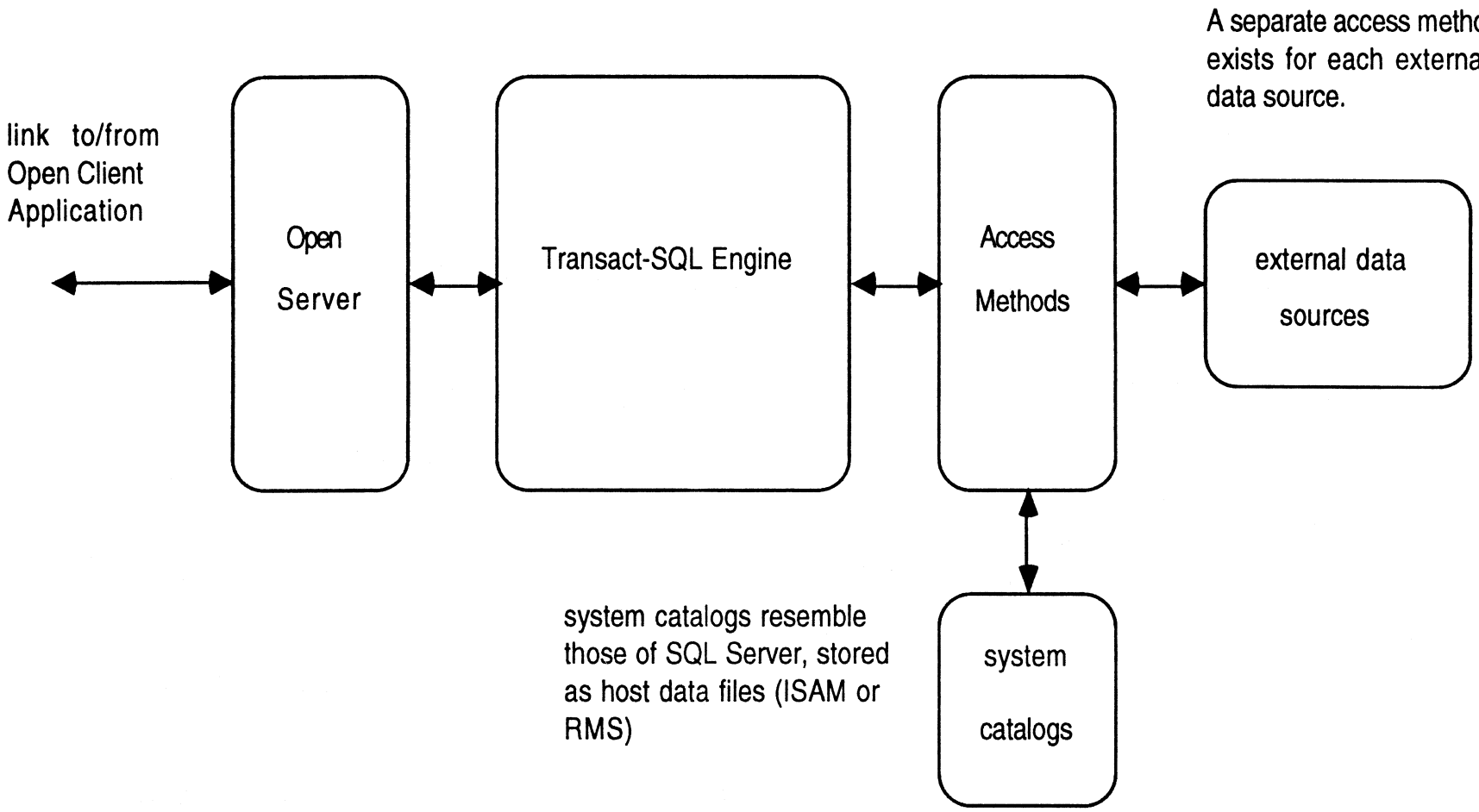


Overview - Omni Architecture

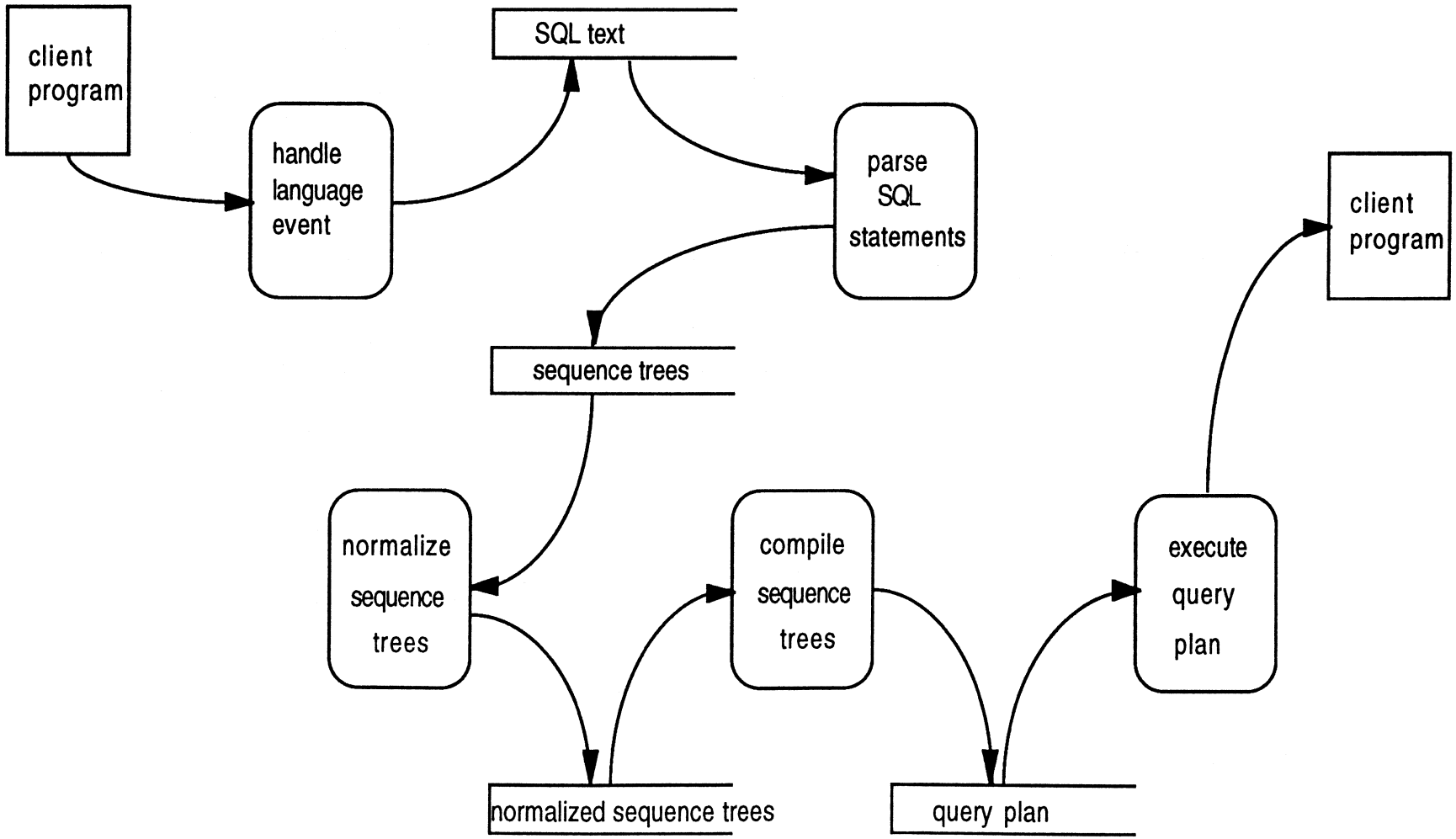
Consists of three primary components:

- Open Server 2.0.1, with EBF's
- Transact-SQL 'Engine' - derived from SQL Server version 4.8:
  - parser
  - sequencer - normalize, compile, execute
- Access Modules - driver for each external data source

Overview - Omni Components



Overview - Language Handling



Overview - Supported Transact-SQL

- Nearly all Transact-SQL commands are supported for all data sources, including:
  - stored procedures
  - views
  - rules
  - defaults
  - triggers (limited capability)
- Read/Write Access to all data sources is provided:
  - select
  - insert
  - delete
  - update

Overview - Supported Transact-SQL

- Transaction Control Statements Supported:
  - begin transaction
  - commit transaction
  - rollback transaction
  - save transaction
  - prepare transaction
- Some limitations exist on different data sources

Overview - Supported Transact-SQL

- Utility Commands Supported:
  - create table
  - create existing table
  - create index
  - drop table
  - drop index
  - alter table
  - truncate table
  - create database
  - drop database

Overview - Supported Transact-SQL

- Miscellaneous Transact-SQL Support:
  - built-in functions
  - local and global variables
  - flow control statements
  - waitfor
  - many others

Overview - Enhanced Transact-SQL

- Enhanced Transact-SQL Support:
  - create existing table
  - create database [on directory]
- New Transact-SQL Support:
  - connect to remote_server_name
  - disconnect

Overview - Unsupported SQL

- Text and image data types
- Compute rows
- Browse mode
- Bulk copy interface
- Two phase commit services
- dump/load database/transaction log
- ALTER DATABASE
- DISK INIT and relatives

Overview - System Catalogs

- Omni's system catalogs are almost equivalent to SQL Server:
  - all but two SQL Server's catalogs are present; most are supported, but some are present for compatibility only
  - two new catalogs are unique to Omni
- On Unix platforms the system catalogs are stored in ISAM files
- On VAX/VMS the system catalogs are stored in RMS files

Overview - Supported Catalogs

- System tables in all databases:
  - sysalternates
  - syscolumns
  - syscomments
  - sysdepends
  - sysindexes
  - syskeys
  - sysobjects
  - sysprocedures
  - sysprotects
  - systypes
  - sysusers

Overview - Supported Catalogs

- System tables in *master* database only:
  - sysconfigures
  - sysdatabases
  - syslanguages
  - syslogins
  - sysmessages
  - sysprocesses
  - sysremotelogins
  - sysservers

Overview - New Catalogs

- System tables in all databases:
  - systabledefs
- System tables in *master* database only:
  - sysexternlogins

Overview - Unsupported Catalogs

- The following system tables exist but are not used:
  - syssegments
  - syscharsets
  - sysdevices
  - syslocks
  - sysusages
- The following system tables do NOT exist:
  - syscurconfigs
  - syslogs

Overview - System Procedures

- Omni supports all SQL Server system procedures, except for the following:

  - sp_addsegment
  - sp_addumpdevice
  - sp_diskdefault
  - sp_dropdevice
  - sp_dropsegment
  - sp_extendsegment
  - sp_helpdevice
  - sp_helplog
  - sp_helpsegment
  - sp_lock
  - sp_logdevice
  - sp_placeobject
  - sp_spaceused

Overview - System Procedures

- The following system procedures are unique to Omni:
  - sp_addexternlogin
  - sp_addtabledef
  - sp_autoconnect
  - sp_dropexternlogin
  - sp_droptabledef
  - sp_fileinfo
  - sp_procio
  - sp_servertype
  - sp_tableio
  - sp_userio

Overview - Possible Uses

- Migration from one RDBMS to another:

```
sp_addtabledef target, "SYBASE.pubs.dbo.target"
go
select * into target from source_table
go
```

When transferring to SQL Server, the bulk copy interface is used internally

Overview - Possible Uses

- Coexistence with Multiple, Heterogeneous Systems:
  - location transparency
  - distributed joins
  - automatic data conversions
  - stored procedures used to encapsulate complex operations
  - rules and defaults are resolved within Omni and applied to all data sources

Overview - Administrative Steps

- Define remote server (data source)

```
sp_addserver 'ORACLE'

sp_servertype ORACLE, oracle6, native,
                "@T:maple:orasid"
```

- Define remote server login

```
sp_addexternlogin ORACLE, sa, system,
                        manager
```

- Should now be able to connect to server in passthru mode:

```
connect to ORACLE
```

Overview - Administrative Steps

- Define remote server tables to be accessed:

```
sp_addtabledef accounts,
                    "ORACLE..system.accounts"

create existing table accounts (
   name      varchar(30),
   id        int,
   balance   money,
   txn_date  datetime,
   etc, etc
)
```

- Grant access privileges to Omni table:

```
grant all on accounts to public
```

Omni Today

- Provided on four platforms:
  - Sun4 SunOS 4.1.2
  - HP9000/800 HP-UX 9.0
  - RS/6000 AIX 3.2
  - VAX/VMS 5.5
  - NCR 3000 SRV4
  - Novell NetWare
- Supports data from:
  - Sybase SQL Server 4.x and above
  - Oracle versions 6, 7
  - DB2 via New Gateway 2.0 and above
  - C-ISAM files (Unix)
  - RMS files (VMS)
  - Ingres
  - Generic access modules

Omni Tomorrow

- Release 10.1 planned for Q2 1994
  - Full System 10 compatibility
    - SQL Server 10.0 code line
    - Uses Open Client/Server 10.0
  - Expanded platform coverage (all core platforms)
  - Full SQL Server datatype support
  - Computed result sets
  - On-going performance enhancements
  - Full internationalization support
  - Extended non-Sybase database coverage via Open Server-based access modules

Omni Tomorrow, continued

- OmniSQL Toolkit
  - Includes full Transact-SQL parser library
  - De-coupled architecture
  - Allows user customizable data access
  - Common module interface to facilitate interchangeable modules
  - Used to develop Omni-compatible Access Modules
  - Consistent Sybase Client-Server Design

- Standard disclaimer:
  - feature set(s) of future releases has not been finalized

Installation - Overview

- Regardless of platform the basic steps are as follows:

  - verify network
  - add sybase user account
  - add SYBASE to environment
  - load Omni software from tape
  - run *omnibuild*
  - create/modify interfaces file
  - run *omniserv*
  - load installmaster
  - load installmodel
  - load installpubs
  - load inscsprocs - ODBC catalog stored procedures

Omni Configuration

- References:

  "OmniSQL Gateway Release Reference Manual for Unix Platforms"

  "OmniSQL Gateway Release Reference Manual for VAX/VMS"

  "System Administration Guide for SYBASE OmniSQL Gateway"

Configuration Overview

- sp_configure configuration variables
- Adding and Configuring remote servers
- Configuring logins **TO** remote servers
- Configuring logins **FROM** remote servers

Configuration - sp_configure

- The use of *sp_configure* is exactly the same as SQL Server, the specific configuration variables differ.

- Omni configuration variables:

allow updates
lock timeout interval (VAX/VMS only)
open databases
open objects
procedure cache size
query buffer size
read regardless (VAX/VMS only)
remote access
remote connections
remote read buffers
remote sites
timer wakeup interval
user connections

Configuration - sp_configure

- Omni configuration variables which operate the **same** as SQL Server:

  allow updates
  open databases
  open objects
  remote access
  remote connections
  remote sites
  user connections

- Omni config variables that **differ** from SQL Server:

  lock timeout interval (VAX/VMS only)
  procedure cache size
  query buffer size
  read regardless (VAX/VMS only)
  remote read buffers
  timer wakeup interval

Configuration - sp_configure

- *lock timeout interval (VAX/VMS only)* configuration variable:

  Omni uses multi-streaming to enable multiple users to access the same files. This variable specifies the number of seconds any stream may wait for a locked RMS record. Zero indicates no waiting.

  Dynamic

  Default = 10 (seconds)

Configuration - sp_configure

- *read regardless (VAX/VMS only)* configuration variable:

  Allows Omni to retrieve RMS records that have been locked exclusively by another process. Enabling this feature will disable *lock timeout interval*, since no waiting for locked records will be done by RMS.

  Dynamic

  Default = 0 (disabled)

Configuration - sp_configure

- *procedure cache size* configuration variable:

  Sets the size of memory, in 2K pages, that Omni allocates from the operating system for use as a cache for stored procedures and as work space when compiling incoming Transact-SQL.

  Default = 500

- *query buffer size* configuration variable:

  Sets the size of memory, in 1K increments, that Omni allocates from the operating system for use as a cache for constructing SQL statements to issue to remote servers.

  Default = 4

Configuration - sp_configure

- *remote read buffers* configuration variable:

  Sets the number of packets which will be preread from remote connections. Default is adequate for virtually all cases.

  Default = 3

- *timer wakeup interval* configuration variable:

  Omni has a background timer which responds to the value of this variable. Primarily affects the effective granularity of the *waitfor* command.

  Default = 5

Configuration - Remote Servers

- In order to access a remote server it must first be defined. This is a multi-step process which consists of *sp_addserver* and *sp_servertype*. Optionally, *sp_serveroption* may be utilized.

- *sp_addserver* functions exactly as the SQL Server. It inserts a row into the sysservers table.

- *sp_servertype* is a system procedure unique to Omni. It is used to define a remote server's class and network access information.

- The valid server classes are:

sql_server	oracle6
oracle7	db2
ingres6	generic

Configuration - sp_servertype

```
sp_servertype server_name, class, interface,
    network_info
```

- server_name - name of server, as found in sysservers, for which a class is to be defined or modified
- class - class of the server. Legal values are *sql_server*, *oracle6, oracle7, ingres6, db2* and *generic.*
- interface - interface type to be used to access the remote server.

  oclient - indicates that open client is going to be used to access the remote server. Valid for *sql_server, db2* and *generic* classes.

  native - indicates that a remote server's native access should be used to access the remote server. Valid for *oracle6, oracle7* and *ingres6.*

Configuration - sp_servertype

```
sp_servertype server_name, class, interface,
    network_info
```

- network_info - varies depending on server class:

sql_server	the name of the remote server as found in the interfaces file.
oracle6/7	SQL*Net address of the Oracle database. Use " " if SQL*Net is not used
db2	Net-Gateway server name, as found in the interfaces file
ingres6	Name of directory that will be used to set the II_SYSTEM environment variable
generic	Generic Access Module, as found in interfaces file

Configuration - sp_servertype Example

```
sp_addserver OMNI, local
go

sp_addserver SYBASE
go

sp_servertype SYBASE, sql_server, oclient, SYB_PUBLIC
go

use master
go

checkpoint
go
```

Configuration - sp_serveroption

- Omni provides two options for remote servers:

  timeouts - works the same as SQL Server.

  writable - unique to Omni. Must be TRUE in order to perform write operations (create, insert, update, delete) on a remote server.

  Allows a remote server to be defined as 'READ ONLY' (if set FALSE).

  Defaults to TRUE for *sql_server* and *oracle6* classes.

  Defaults to FALSE for *db2* class.

Configuration - helpserver & dropserver

- Omni also provides both *sp_helpserver* and *sp_dropserver*. Each of these procedures operates the same as on a SQL Server.

Configuration - sp_addexternlogin

- *sp_addexternlogin* is used to instruct Omni how to log a given Omni user into a remote server.
- Inserts or updates a row in *sysexternlogins*

```
sp_addexternlogin server, loginname, externname,
    externpasswd
```

- *server* - name of remote server as found in sysservers
- *loginname* - name of Omni user as found in syslogins
- *externname* - login name to use with remote server
- *externpasswd* - login password to use with remote server

Configuration - sp_addremotelogin

- *sp_addremotelogin* is used to instruct Omni which remote SQL Server users can access the server, and what identity they will assume.
- *sp_addremotelogin* works the same as SQL Server.

Server Classes - Overview

- Server class *local*
- Server class *sql_server*
- Server class *oracel6, oracle7*
- Server class *db2*
- Server class *generic*
- Passthrough mode
- The *defgen* utility

Server Class - local

- The local server is the currently running Omni server
- Objects managed by the local server include host data files
  - on Unix, these are ISAM files
  - on VMS, these are RMS files
- Unless table mapping has been specified, it is assumed that the table about to be created is a *local* object
- Default mapping for local tables can be overridden using the *sp_addtabledef* procedure:

Server Class - local

```
sp_addtabledef table_name, "pathname"
```

- pathname can be any legal file specification:
  - read access is required by the Omni process
- If an existing object is an ISAM file, don't use the file's suffix in the pathname (.idx or .dat)
- During the *create existing* function, file attributes are obtained from the file system and used to update Omni's system catalogs.

Server Class - local - Unix systems

- Datatype mapping for *create table* ; Omni type (left) maps to type used by C-ISAM (right):

binary(n)	chartype(n)
bit	chartype(1)
char(n)	chartype(n)
datetime	2 longtype fields
float	doubletype
int	longtype
money	longtype, chartype(4)
real	floattype
smalldatetime	chartype(4)
smallint	inttype
smallmoney	longtype
timestamp	longtype
tinyint	chartype(1)
varbinary(n)	chartype(n)
varchar(n)	varchar(n)

Server Class - local - Unix systems

- Datatype mapping for *create existing table* ; ISAM type (left) maps to type used by Omni (right):

chartype(n)	binary(n)
chartype(n)	varbinary(n)
chartype(n)	char(n)
chartype(n)	varchar(n)
chartype(1)	tinyint
decimal	binary(n) (application must convert)
doubletype	float
floattype	real
inttype	smallint
longtype	int

- Type checking is not performed, you must know the layout of the ISAM records

Server Class - local - VMS systems

- Datatype mapping for *create table* ; Omni type (left) maps to type used by RMS (right):

binary(n)	string(n)
bit	string(1)
char(n)	string(n)
datetime	VAX date (quadword)
float	D-float
int	longword
money	quadword
real	float
smalldatetime	longword
smallint	word
smallmoney	word
timestamp	longword
tinyint	string(1)
varbinary(n)	string(n)
varchar(n)	string(n)

Server Class - local - VMS systems

- Datatype mapping for *create existing table* ; RMS type (left) maps to type used by Omni (right):

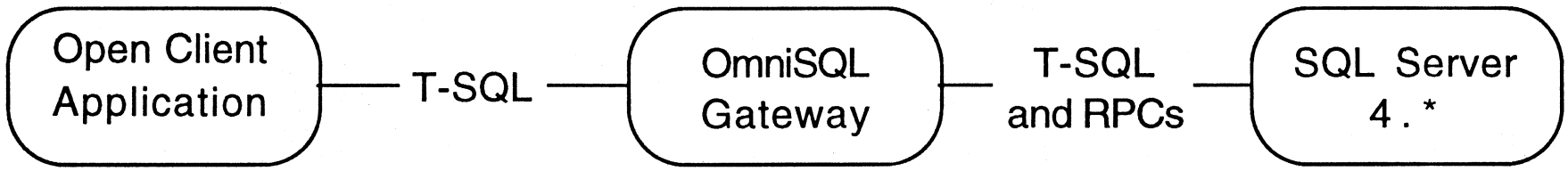
string(n)	binary(n)
string(n)	varbinary(n)
string(n)	char(n)
string(n)	varchar(n)
string(1)	tinyint
packed decimal	binary(n) (application must convert)
D-float	float
G-float	binary(8) (application must convert)
VAX date	datetime
word	smallint
longword	int
quadword (VAX date)	datetime

- Type checking is not performed, you must know the layout of the RMS records

- Column offsets and length are checked for match when key descriptor is found

Server Class - sql_server

Client uses Open Client interface to communicate with Omni, sending only Transact-SQL.



OmniSQL uses Open Client interface to communicate with SQL Servers, sending only T-SQL and RPCs.

Server Class - sql_server - data mapping

- Datatype mapping for *create table* and *alter table*; Omni type (left) maps to type used by SQL Server (right):

binary	binary
bit	bit
char	char
datetime	datetime
float	float
int	int
money	money
real	real
smalldatetime	smalldatetime
smallint	smallint
smallmoney	smallmoney
timestamp	timestamp
tinyint	tinyint
varbinary	varbinary
varchar	varchar

Server Class - sql_server - data mapping

- Datatype mapping for *create existing table*; SQL Server type (left) maps to type allowed by Omni (right):

binary	binary
bit	bit
char	char
datetime	datetime
float	float
image	UNSUPPORTED
int	int
money	money
real	real
smalldatetime	smalldatetime
smallint	smallint
smallmoney	smallmoney
text	UNSUPPORTED
timestamp	timestamp
tinyint	tinyint
varbinary	varbinary
varchar	varchar

- System 10 data types not yet supported

Server Class - sql_server - server definition

- Server definition via *sp_servertype*:

  server_name - local Omni server name

  class - always *sql_server* for Sybase servers

  interface - always *oclient*

  network_info - name of server as found in *interfaces* file

- No separate class yet for System 10 SQL Server

Server Class - sql_server - table mapping

- Mapping done via *sp_addtabledef*:

  table_name - local Omni table name

  remote_name - path to remote table or view in the form:

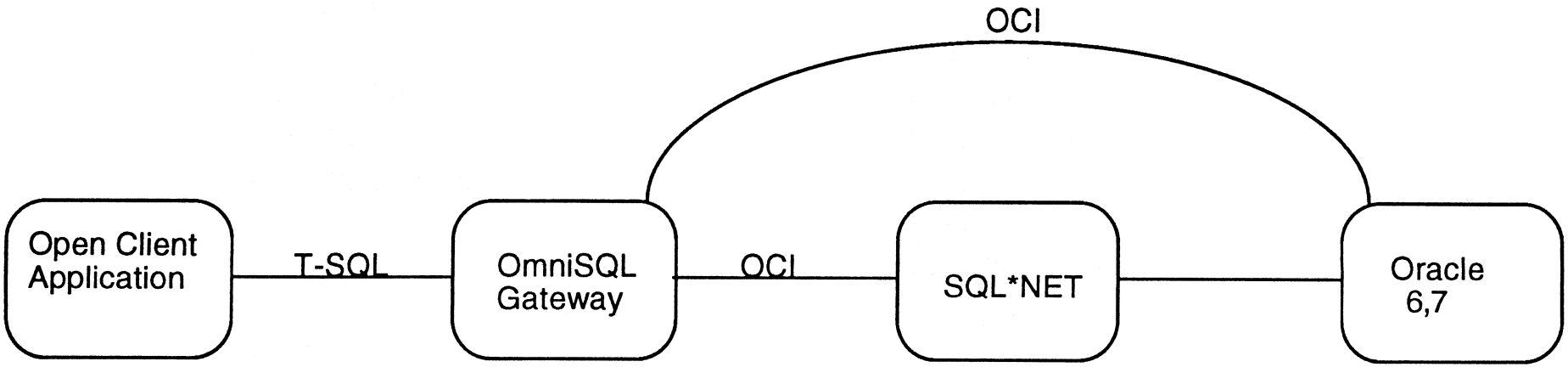
  server.database.owner.object

  where:

  server - name of Omni server
  database - name of database in remote SQL Server
  owner - name of owner of remote object
  object - name of either table or view in server

Server Class - oracle6, 7

Client uses Open Client interface to communicate with Omni, sending only Transact-SQL



OmniSQL uses Oracle's OCI interface to communicate with Oracle, either directly or via SQL*NET

Server Class - oracle6, 7 - data mapping

- Datatype mapping for *create table* or *alter table*; supplied Omni type (left) maps to type used by Oracle (right):

binary	raw
bit	number(1)
char	char
datetime	date
float	float
int	number(10)
money	number(19,4)
real	float
smalldatetime	date
smallint	number(5)
smallmoney	number(19,4)
tinyint	number(3)
varbinary	raw
varchar	char

Server Class - oracle6, 7 - data mapping

- Datatype mapping for *create existing table*; Oracle type (left) maps to type allowed by Omni (right):

char	char, varchar
rowid	char, varchar
number	bit
number	tinyint, smallint, int
number	real, float
number	smallmoney, money
date	smalldatetime, datetime
long	varchar(255)
raw	binary
longraw	varbinary(255)
varchar	varchar(255)

Server Class- oracle6,7- data mapping, passthru

- Datatype mapping for *passthrough mode*; Oracle type (left) maps to Omni type (right):

char	varchar
rowid	varchar
number	float
date	datetime
long	varchar(255); truncated if length > 255
raw	binary
longraw	varbinary(255); truncated if length > 255
varchar(oracle7)	varchar(255); truncated if length > 255

Server Class - oracle6, 7 - server definition

- Server definition via *sp_servertype*:

  server_name - local Omni server name

  class - either *oracle6 or oracle7* for Oracle servers

  interface - always *native*

  network_info - SQL*Net address of Oracle database, or ""

Server Class - oracle6, 7 - SQL*Net definition

- Network information provided by *sp_servertype* can specify SQL*Net connect string for local or remote Oracle servers

- If Oracle and Omni are on the same system, SQL*Net is not required. Use empty string ("") and set Oracle environment variables ORA_SID and ORA_SYSTEM before starting Omni

- SQL*Net string can specify either TCP/IP or DECNet protocols to be used:

  - TCP/IP:
  @T:host_name:ora_sid[,retry_cnt,buf_size,keepalive]

  - DECNet:
  @D:host_name-ORDNora_sid OR

  @D:host"name passwd"::"TASK=filename"

Server Class - oracle6, 7 - table mapping

- Mapping done via *sp_addtabledef*:

  table_name - local Omni table name

  remote_name - path to remote table or view in the form:

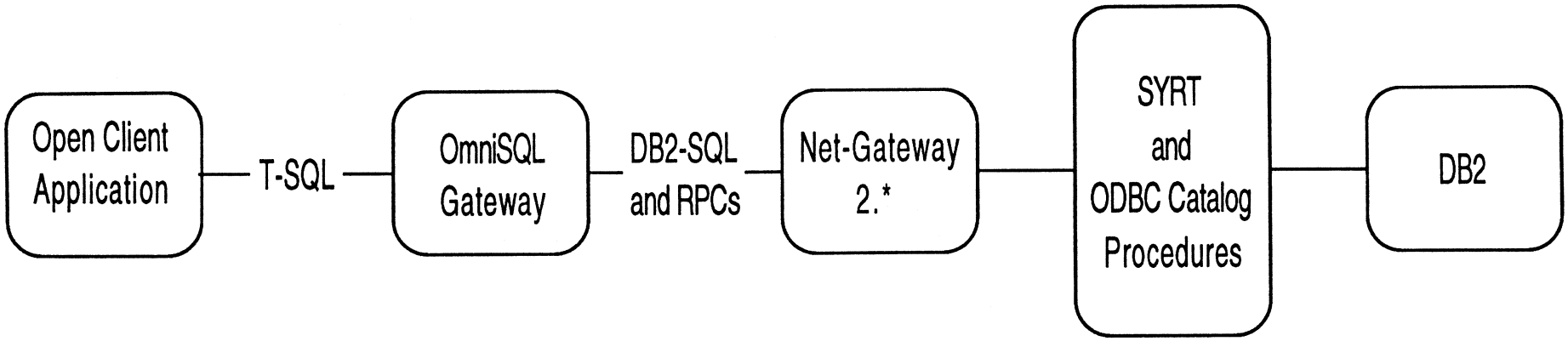
  server.database.owner.object

  where:

  server - name of Omni server
  database - ignored, since no Oracle support for databases
  owner - name of owner of remote object
  object - name of either table or view in Oracle

Server Class - db2

Client uses Open Client interface to communicate with Omni, sending only Transact-SQL.



OmniSQL uses Open Client interface to communicate with Net-Gateway, sending only DB2 and RPCs.

Server Class - db2 - data mapping

- Datatype mapping for *create table* and *alter table*; supplied Omni type (left) maps to type used by DB2 (right):

binary(n)	varchar(n*2); n <= 127
bit	char(1)
char(n)	char(n); n <= 254
datetime	timestamp
float	float
int	int
money	float
real	real
smalldatetime	timestamp
smallint	smallint
smallmoney	float
tinyint	smallint
varbinary(n)	varchar(n*2); n <= 127
varchar(n)	varchar(n); n <= 254

Server Class - db2 - data mapping

- Datatype mapping for *create existing table*; DB2 type (left) maps to type allowed by Omni (right):

int	int
smallint	smallint
float(n); n <= 21	real, float, money
float(n); n > 21	float, money
float	float, money
double precision	float, money
real	real, float, money
decimal; scale > 0	float, money
decimal; scale = 0	int, float, money
numeric; scale > 0	float, money
numeric; scale = 0	int, float, money
char	char, varchar
varchar	char, varchar
long varchar	char(255), varchar(255)

Server Class - db2 - data mapping

- Datatype mapping for *create existing table,* continued: DB2 type (left) maps to type allowed by Omni (right):

date	char, varchar, datetime (time=12:00am)
time	char, varchar, datetime (date=1/1/1900)
timestamp	char, varchar, datetime
tinyint	tinyint
graphic	UNSUPPORTED
vargraphic	UNSUPPORTED
long vargraphic	UNSUPPORTED

Server Class - db2 - server definition

- Server definition via *sp_servertype*:

  server_name - local Omni server name

  class - always *sql_server* for Sybase servers

  interface - always *oclient*

  network_info - name of Sybase Net Gateway as found in *interfaces* file

- Net Gateway version 2.0 is required

Server Class - db2 - table mapping

- Mapping done via *sp_addtabledef*:

  table_name - local Omni table name

  remote_name - path to remote table or view in the form:

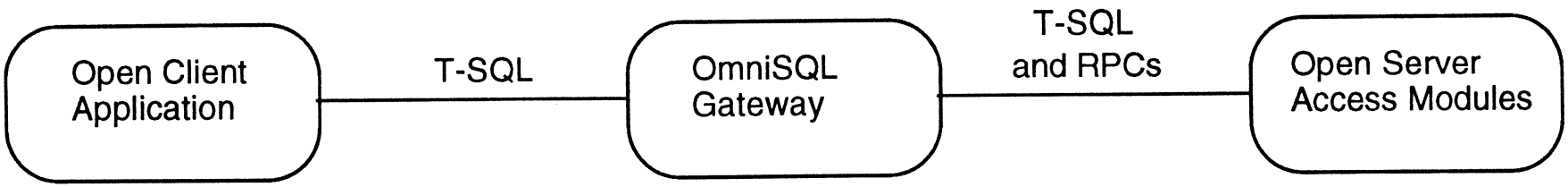
  server.database.owner.object[;aux1,aux2]

  where:

  server - name of Omni server
  database - location-name portion of DB2 table name
  owner - DB2 authorization id
  object - name of either table or view in server
  aux1 - DB2 database in which locate table
  aux2 - DB2 tablespace in which to locate table

Server Class - generic

Client uses Open Client interface to communicate with Omni, sending only Transact-SQL



OmniSQL uses Open Client interface to communicate with Open Server modules, sending only T-SQL and RPCs

Server Class - generic - data mapping

- Datatype mapping for *create table* and *alter table*; Omni type (left) maps to type used by generic Server (right):

binary	binary
bit	bit
char	char
datetime	datetime
float	float
int	int
money	money
real	real
smalldatetime	smalldatetime
smallint	smallint
smallmoney	smallmoney
timestamp	timestamp
tinyint	tinyint
varbinary	varbinary
varchar	varchar

Server Class - generic - data mapping

- Datatype mapping for *create existing table*; generic Server type (left) maps to type allowed by Omni (right):

binary	binary
bit	bit
char	char
datetime	datetime
float	float
image	UNSUPPORTED
int	int
money	money
real	real
smalldatetime	smalldatetime
smallint	smallint
smallmoney	smallmoney
text	UNSUPPORTED
timestamp	timestamp
tinyint	tinyint
varbinary	varbinary
varchar	varchar

- System 10 data types not yet supported

Server Class - generic - server definition

- Server definition via *sp_servertype*:

  server_name - local Omni server name

  class - use *generic* for Generic Access Modules

  interface - always *oclient*

  network_info - name of server as found in *interfaces* file

Server Class - generic - table mapping

- Mapping done via *sp_addtabledef*:

  table_name - local Omni table name

  remote_name - path to remote table or view in the form:

  server.database.owner.object

  where:

  server - name of Omni server
  database - name of database in Generic Access Module (optional)
  owner - name of owner of remote object
  object - name of either table or view in server

Server Classes - Passthrough Mode

- Passthrough mode allows an Omni client to interact directly with a remote server. Once in passthrough mode, Omni simply 'passes through' client requests (without translation) and, in return, 'passes through' server result sets.

- To enter passthrough mode:

```
connect to server_name
```

- *server_name* - name of server as found in sysservers

- To exit passthrough mode:

```
disconnect
```

Server Classes - sp_autoconnect

- *sp_autoconnect* places new user into passthru mode whenever user logs in to Omni:

```
sp_autoconnect server_name, true | false,
    [ login_name ]
```

- *server_name* - name of server as found in sysservers
- *true* | *false* - "true" enables autoconnect mode; "false" disables it
- *login_name* - name of user; can only be used by 'sa'; defaults to current user name.

Server Classes - *defgen* Utility

- Provides quick and easy definition of remote tables to Omni.
- Generates SQL script file which contains *sp_addtabledef* and *create existing table* statements for tables in remote server.
- Remote tables can be specified individually, by owner, by database or by server.
- Remote server name and class must be defined in Omni first.
- Default data type conversions are used; the script file can be edited to override defaults.

Server Classes - *defgen* example

- Define all tables owned by 'dbo' in database 'pubs' in remote SQL Server:

```
defgen -Usa -P -SOMNI -Dtestdb -sSYBASE -dpubs
       -ndbo -Fsybpubs.sql
```

- Define all tables owned by user 'scott' in remote Oracle server:

```
defgen -Usa -P -SOMNI -Dtestdb -sORACLE -nscott
       -Fora.sql
```

- Define selected tables in remote DB2 server:

```
defgen -Usa -P -SOMNI -Ddb2_tables -sDB2
       -Fora.sql dsn8220.act dsn8220.emp
```

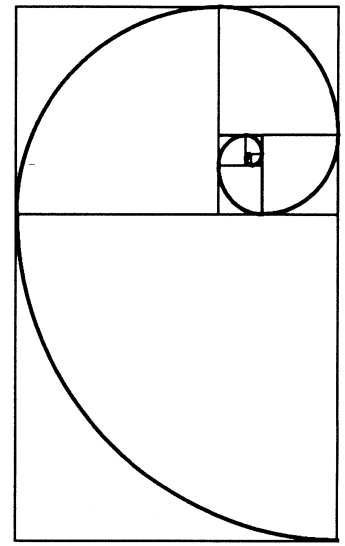
Server Classes - *defgen* example

- After *defgen* finishes, output file can be edited, if necessary, to modify default data type mapping
- Output file is then passed to Omni for execution:

```
isql -Usa -P -SOMNI < sybpubs.sql >sybputs.out
```

Support Issues

- If Omni is failing to connect to a remote server then try connecting direct to the remote server (without Omni).
- Remote tables can be altered outside of Omni. Hence, undefined results can occur if Omni's knowledge of external tables is not kept up to date.
- Permissions set via the *grant* statement are only local to Omni.
- Omni (current user) needs enough permissions to query remote catalog tables in order to perform a *create existing* command.



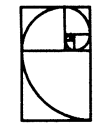
Sybase

System 10 SQL Server
ANSI Standards & IEF

Author: Manish Chandra

ANSI 89 & SQL 92 Features

- **FULL ANSI 89/FIPS 127-1 COMPLIANCE IN THE SYSTEM 10 SQL SERVER.**
- **FEATURES TARGETTING ENTRY LEVEL SQL 92.**



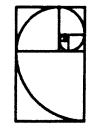
Transaction Isolation Levels

- Users can set transaction isolation levels for the entire session.
- Pre 10.0 way of doing level 3 isolation:

```
BEGIN TRAN
SELECT Name, Id FROM EMPS HOLDLOCK
SELECT Salary FROM SalTable HOLDLOCK
.....
```

- System 10 SQL Server allows setting of isolation level for the entire session

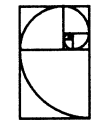
```
SET TRANSACTION ISOLATION LEVEL 3
BEGIN TRAN
SELECT Name, Id FROM EMPS
SELECT Salary FROM SalTable
.....
```

Chained and Unchained Transactions

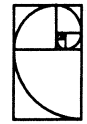
- ANSI requires all selects and DML statements to be executed in a transaction.
- A new SET command allows applications to switch to the ANSI Transaction/Chained Mode.

  SET CHAINED ON



Numeric Datatypes

- System 10 SQL Server supports the NUMERIC & DECIMAL datatypes with user specified precision and scale.
- The Numeric columns can store very large numbers (upto 38 digits).
- Any loss of information is flagged. A truncation warning is raised to the application. A SET option is provided to turn off the warning.
- All Float literals without an exponent are parsed as Numerics giving higher accuracy in computations.



Numeric Datatypes

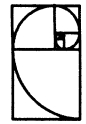
- Example:

  SELECT 1234563.123433432432143324 32
  would return the following in pre 10.0 SQL Servers:
  1234563.123433

  In 10.0 SQL Server, all information is preserved:
  1234563.123433432432143324 32

- Columns can have user specified precision and scale:

  CREATE TABLE Accounts(AcctId NUMERIC(10, 0),
  AcctBalance NUMERIC(32, 8))

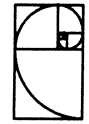


Views with Check Option

- A new way of enforcing integrity of data.
- Prevents users from changing the data visible through a view in such a way that the data is no longer visible through the view.
- Where can this be used?

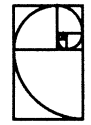
```
CREATE VIEW usr1_view
AS
SELECT Name, Inventory, Warehouse
FROM Table1
WHERE Warehouse IN ("Los Angeles", "San Francisco", "Paris")
WITH CHECK OPTION
```

- The user can not change the table Table1 so that Warehouse is outside the range shown in the WHERE clause. This means that usr1 can't move inventory to a place outside his control.



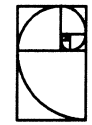
Declarative Integrity (IEF)

- PRIMARY KEY can be declared on a table. This enforces non-null uniqueness on a group of columns.
- FOREIGN KEY relationship can be declared between two tables.
- FOREIGN KEY relationship ensures that there are no detailed records present without a corresponding primary record.
- FOREIGN KEY relationship is blocking and non-cascading.



Declarative Integrity (IEF)

- UNIQUE constraints can be used to enforce uniqueness of a group of columns. Multiple UNIQUE constraints can exist on a single table.
- DEFAULT constraint can be used to supply defaults for the columns.
- CHECK constraint can be used to enforce certain conditions on a group of columns in the table.

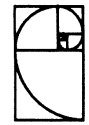


Declarative Integrity (IEF)

- The messages printed out on integrity violations can be customized for different applications using a new system stored procedure:

  sp_bindmsg constraint_name, message_#

- The integrity constraints can be added or removed from the tables dynamically using the ALTER TABLE commands.
- The new declarative constraints can be used with existing RULEs, TRIGGERs and DEFAULTs.



Declarative Integrity (IEF)

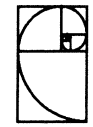
EXAMPLE APPLICATION

- The task is to set up a schema for the following set of tables which store the payroll information of a set of employees in the Big Benevolent Company.

  Departments - Each department has an id, name and a location.

  Employees - Each employee has an id, name, address and belongs to a department.

  Salaries - For each employee this table stores the current salary level, current pay grade and the previous salary level.



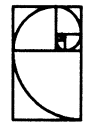
Declarative Integrity (IEF)

EXAMPLE APPLICATION

- The following set of constraints must be imposed on the data stored in the tables:

  1. No two employees should have the same id.
  2. No two departments should have the same id.
  3. Each employee belongs to a valid department.
  4. No employee in the company should make less than $100,000.
  5. The minimum pay raise given to any employee is 20%.

  Remember, it is the Big Benevolent Company!

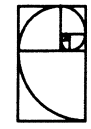


Declarative Integrity (IEF)

EXAMPLE APPLICATION

- In pre 10.0 SQL Server the following is the schema with some triggers and rules which will have to be written to ensure that the above constraints are met.

```
CREATE TABLE depts(
        deptid CHAR(6),
        dept_name CHAR(30))
CREATE UNIQUE INDEX dept_idx1 ON depts(deptid)
CREATE TABLE emps(
        empid NUMERIC(8, 0),
        emp_name CHAR(30),
        emp_addr CHAR(60),
        dept_id CHAR(6))
CREATE UNIQUE INDEX emp_idx1 ON emps(empid)
```

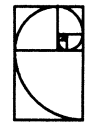



Declarative Integrity (IEF)

EXAMPLE APPLICATION

```
CREATE TABLE salaries (empid NUMERIC(8, 0),
                       salary NUMERIC(12, 2),
                       prev_sal NUMERIC(12, 2),
                       pay_grade CHAR(2))
```

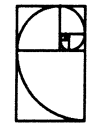
```
/* Now write a trigger on table employees to ensure that there
** is a valid department id when we insert a records in the
** employee table. This will ensure one half of the primary
** key - foreign key relationship between emps and depts.
*/
CREATE TRIGGER emp_insupdtrig
ON emps
FOR INSERT, UPDATE
AS /* contd on the next slide */
```

Declarative Integrity (IEF)

EXAMPLE APPLICATION

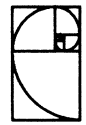
```
IF UPDATE(deptid)
BEGIN
        IF EXISTS (SELECT * FROM emps E
                WHERE deptid IS NOT NULL
                AND NOT EXISTS(SELECT *
                    FROM depts D
                    WHERE D.deptid = E.deptid)
        BEGIN
            ROLLBACK TRANSACTION
            PRINT "No Primary Key Exists in
                table depts for this record"
        END
END
```

Declarative Integrity (IEF)

EXAMPLE APPLICATION

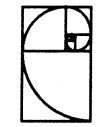
1. A trigger will have to be put on each table to enforce the primary key-foreign key relationship.
2. After that we'll have to write rules to enforce the salary constraints on this schema.
3. Overall the set of triggers and rules for these kinds of constraints can become cumbersome.



Declarative Integrity (IEF)

EXAMPLE APPLICATION - SYSTEM 10 SOLUTION

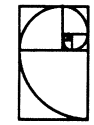
```
CREATE TABLE depts(
        deptid CHAR(6) PRIMARY KEY,
        dept_name CHAR(30))

CREATE TABLE emps(
        empid NUMERIC(8, 0) PRIMARY KEY,
        emp_name CHAR(30),
        emp_addr CHAR(60),
        dept_id CHAR(6) REFERENCES depts(deptid))
```

Declarative Integrity (IEF)

EXAMPLE APPLICATION - SYSTEM 10 SOLUTION

```
CREATE TABLE salaries (
                empid NUMERIC(8, 0)
    REFERENCES emps(empid),
                salary NUMERIC(12, 2),
                prev_sal NUMERIC(12, 2),
                pay_grade CHAR(2),
    CONSTRAINT min_raise
            CHECK ((salary - prev_sal)/prev_sal >= 0.20),
    CONSTRAINT min_sal
            CHECK(salary > $100,000))
```

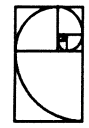



Declarative Integrity (IEF)

- The declarative integrity does simplify enforcement of certain kinds of constraints.
- It, however, does not completely replace Triggers and Rules.

  An example of a constraint which IEF can not simulate easily:

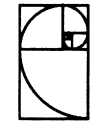
  If instead of ensuring all employees salary levels to be above $100, 000, the Big Benevolent Company decided that only the employees working in the MIS department should earn more than $100,000.



Declarative Integrity (IEF)

- A new system stored procedure **sp_helpconstraint** will display information about the various constraints on a table.
- Example:

```
> sp_helpconstraint Table1
 name         defn
 ------       ------------------------------------------
 def1
     DEFAULT "Wilt"
 prim
     PRIMARY KEY INDEX (a):CLUSTERED
 cand
     UNIQUE INDEX (b, c) NONCLUSTERED
 ref
     foo FOREIGN KEY (c) REFERENCES zeno(z)
 check1
     CHECK (a > 0)
```

Grant with Grant

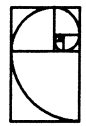
- The System 10 SQL Server provides the capability to grant privileges to the users in such a way that the users can subsequently grant these privileges to other users.
- Example:

  If the table owner issues the following command:
  GRANT SELECT ON table1 TO Jerry WITH GRANT OPTION

  Jerry can in turn, now grant this privilege to another user.

  GRANT SELECT ON table1 TO Mona

- REVOKE revokes only the privileges granted by the user and does not effect the GRANTs by the other users.



Grant with Grant

- The users should be aware of the following GRANT WITH GRANT behavior:

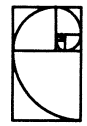
  Table Owner GRANTs SELECT privilege on Table1 to User1 WITH GRANT OPTION.

  Table Owner GRANTs SELECT privilege on Table1 to User2.

  User1 GRANTs the SELECT privilege on Table1 to User2 and User3.

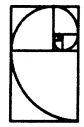
  Table Owner REVOKEs SELECT on Table1 from User2.

  User2 will still have the SELECT privilege that User1 GRANTed on Table1 to it.



Grant with Grant

- In System 10 SQL Server all GRANTs are tracked based on who granted the privilege and only a matching revoke can revoke the privilege.
- However, in the previous example, if the Table Owner had also REVOKEd the SELECT privilege on Table1 from User1 the privilege would have been automatically REVOKEd from User2 and User3.
- Now User2 has no SELECT privilege on Table1.



Features required for Entry Level SQL 92

- Escaping the key words using quotes

  A new SET option is used to tell the parser that quotes are being used to escape keywords

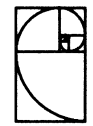
  SET QUOTED_IDENTIFIER ON

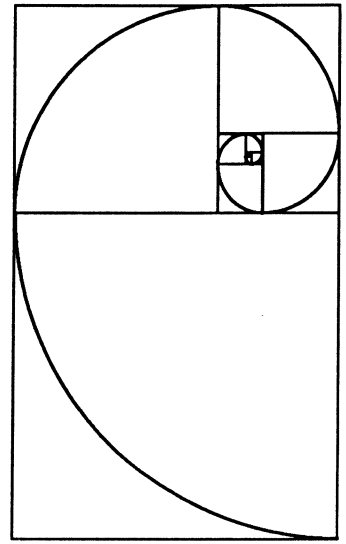
  After setting the option on, the user can issue the following DDL:
  CREATE TABLE "$%garBAGE" ("UPDATE" int,
  "SELECT" char)

- SQLSTATE

  This is an information string which divides errors into classifications based on their characteristics.

- Support for the AS clause in the SELECT.





Sybase

System 10 SQL Server

Database Cursors

Author: Manish Chandra

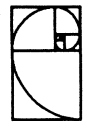
Database Cursors

- What is a Database Cursor?

  It is a mechanism for accessing the results of a SQL SELECT statement, one row at a time.

  Normally, a SELECT will return all the rows to the application in some order. Currently DB-Lib provides a mechanism to process each row. But the buffering is done purely by the client.

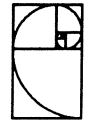
  With the System 10 SQL Server, the rows are returned to the application on demand rather than all at once.



Database Cursors

- Using the Cursor mechanism the applications can take action on each row rather than the entire set of rows returned by the SELECT.
- The applications can do Positioned Updates and Deletes.
- The cursor can be thought of as a handle on the result of a SELECT statement e.g

```
/* The following statement associates/declares a cursor on
** a SELECT statement.
*/
DECLARE test_curs CURSOR
FOR
SELECT name, title
FROM employees
```

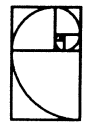



Database Cursors

- After a cursor has been attached to a SELECT statement, it has to be 'OPEN'. This means that the server should start processing this statement.

```
OPEN test_curs
```

- System 10 cursors optimize the processing at OPEN time. In most cases only partial processing is done at this time. This means that the entire result set is not generated at this point in time.



Database Cursors

- The application can then access each row in the SELECT's result set by using the FETCH operation.

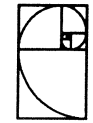
  FETCH test_curs

  This will return one row at a time.

- The results of a FETCH can be stored in variables/ parameters i.e

  FETCH test_curs INTO @name, @title

- The status of the FETCH is stored in a global variable called @@sqlstatus. By checking the value of this variable the applications can quickly tell whether the FETCH was successful or not.

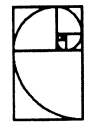


Database Cursors

- If during processing the application wants to change a particular row, a positioned update can be issued. Let's say we want to change the title of a particular employee.

```
UPDATE employees
SET title = "Small Fry"
WHERE CURRENT of test_curs
```

- The cursor scan position is not changed and the scan can continue from this point.



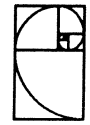
Database Cursors

- After processing as many rows as the application needs, the cursor can be closed. The closing of a cursor frees up valuable system resources and also resets the scan. Before using the cursor again an OPEN must be issued.

  CLOSE test_curs

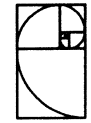
- System 10 SQL Server also provides another command DEALLOCATE to completely clean up the resources of a cursor. After issuing this command the cursor must be redeclared before it can be used.

  DEALLOCATE CURSOR test_curs



Database Cursors

- Cursors can be declared through the System 10 Precompilers and CT-library.
- Cursors can be declared in T-SQL, stored procedures and triggers.
- Cursors can be declared on the results of a stored procedure which contains a single SELECT statement. This can only be done through the pre-compiler or CT-library.

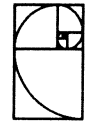


Database Cursors

- Cursors can remain open across transactions.
- A new SET option determines the effect of the transaction end on the open cursors.

  SET CLOSE ON ENDTRAN ON

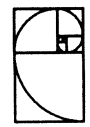
- Leaving cursors open across transactions provides higher concurrency and performance.



Database Cursors

EXAMPLE APPLICATION

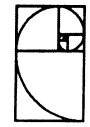
- Design a stored procedure which does the monthly closing of bank accounts for The Small Community Bank.
- If the balance in the Checking account exceeds $1000 then an interest is added to the account.
- If the balance exceeds $10,000 then a bonus is also given to the account.
- However, if the balance is below $500 then a service charge is deducted from the account.
- The service charge, bonus and interest rates can all vary from month to month.



Database Cursors

- Let us first write the procedure without using database cursors.

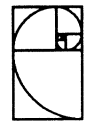
```
/* The following procedure closes the accounts at the end of
** every month.
*/
CREATE PROCEDURE close_accts(
                    @int_rate NUMERIC(6, 6),
                    @serv_chg NUMERIC(8, 6),
                    @bonus NUMERIC(8, 6))
AS
/* Start a reconciling transaction */
BEGIN TRANSACTION
/* First update the accounts which are above $1000 */
UPDATE accounts
SET balance = balance + (@int_rate * balance)
WHERE balance >= $1000
```

Database Cursors

```
/* Next update the accounts which meet the bonus criterion */
UPDATE accounts
SET balance = balance + @bonus
WHERE balance >= $10000

/* Now deduct the service charge */
UPDATE accounts
SET balance = balance - @serv_chg
WHERE balance < $500

COMMIT TRANSACTION

RETURN
```

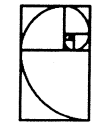
NOTE: This procedure scans the table three times.



Database Cursors

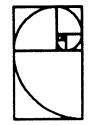
- Now let's rewrite the procedure using the database cursors.

```
/* The following procedure closes the accounts at the end of
** every month.
*/
CREATE PROCEDURE close_accts(
                    @int_rate NUMERIC(6, 6),
                    @serv_chg NUMERIC(8, 6),
                    @bonus NUMERIC(8, 6))
AS
DECLARE     @doupdate int
DECLARE     @balance   NUMERIC(20, 6)
/* Now declare a cursor on the SELECT from accounts */
DECLARE curs CURSOR
FOR
SELECT balance
FROM accounts
FOR UPDATE OF balance
```

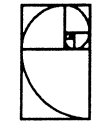
Database Cursors

```
/* Next open the cursor and start the processing */
OPEN curs
/* fetch the first row */
FETCH curs INTO @balance
/* Now loop processing all the rows.
** @@sqlstatus = 1 - means error on previous fetch.
** @@sqlstatus = 2 - means end of result set reached.
*/
WHILE (@@sqlstatus!= 2)
BEGIN
    /* Always check for any errors first. */
    IF (@@sqlstatus = 1)
    BEGIN
            EXEC error_handler
            RETURN
    END
```

Database Cursors

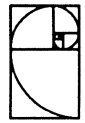
```
/* Next compare the balance against different values */
IF (@balance < $500)
BEGIN
    SELECT @balance = @balance - @serv_chg
    SELECT @doupdate = 1
END
ELSE IF (@balance >= $1000)
BEGIN
    SELECT @balance = @balance * (1+@int_rate)
    IF (@balance >= $10000)
        SELECT @balance = @balance + @bonus
    SELECT @doupdate = 1
END
ELSE
BEGIN
    SELECT @doupdate = 0
END
```

Database Cursors

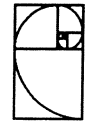
```
        IF (@doupdate = 1)
        BEGIN
            /* Next update the account with the new balance */
            UPDATE accounts
            SET balance = @balance
            WHERE CURRENT OF curs
        END

        FETCH curs INTO @balance
    END

    CLOSE curs
    RETURN
```

- NOTE: Each update commits as it is made.



Database Cursors

- We avoided multiple scans of the table.
- The second procedure used shorter transactions.
- Each row was touched once and kept locked for less time. This provides more concurrency and higher throughput.



Database Cursors

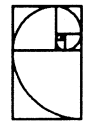
- Cursors can be declared to be read-only or updatable.

  This distinction helps the SQL Server in optimizing the queries.

  It can be used to influence the locking behavior of the cursors. A new kind of lock is introduced, called the Update Lock, to provide the stability needed for updatable cursors.

  It helps in making the application programs both more readable and maintainable.

```
DECLARE curs CURSOR
FOR
SELECT name, title
FROM employees
FOR UPDATE OF title
```

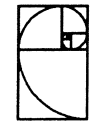
Database Cursors-Hints for Development

- Cursors and row positions

  The SQL Server dynamically maintains tables and indexes. This means that during a cursor scan, other updates by the same process to these tables can interfere with the scan.

  The SQL Server uses unique indexes for scanning the tables to avoid these problems. An example of such a problem is a table with no clustered index where if a row changes size it migrates to the end of the table.

  In any case, the columns which make up the index being used to access the tables should never be changed or else the rows can reappear or disappear.

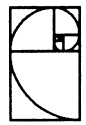


Database Cursors-Hints for Development

- Cursors and Identity

  The new Identity property columns can be used to influence the cursor's selection of certain access paths for tables.

  Any index which includes the identity property column is implicitly available for cursor's access paths.



Database Cursors & the new locking mode

- Update Locking and Cursors:

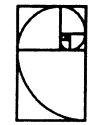
  A table can be locked using Update locking mode when being accessed through a cursors.

  The update locks have the following properties:

  Update Lock conflicts with another Update Lock and Exclusive Lock.

  Update Lock is compatible with other Shared Locks.

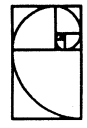
- This means that after a Fetch, the scanner is assured that no other Fetch with an intention to update can be done on that - page. This may result in fewer deadlocks. However, this may also reduce concurrency.



Database Cursors- Another example

- Example: Create a trigger on a salary table which checks to see:

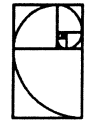
  1. Whether the new salary level is less than the salary level of the employee's supervisor.
  2. It should also check that the new salary level is not more than the maximum salary level specified for that department.



Database Cursors- Another example

- The schema for the example is shown below:

```
CREATE TABLE depts(deptid NUMERIC(5, 0) PRIMARY
KEY, deptname CHAR(40), max_salary NUMERIC(15, 4))

CREATE TABLE employees(empid NUMERIC(10, 0)
IDENTITY PRIMARY KEY, manager NUMERIC(10, 0),
empname CHAR(30), deptid NUMERIC(5, 0)
REFERENCES depts)

CREATE TABLE salary(empid NUMERIC(10, 0) UNIQUE
REFERENCES employees(empid), salary_level
NUMERIC(15, 4))
```

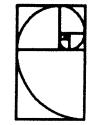



Database Cursors- Another example

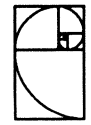
- The following trigger uses cursors on inserted and deleted tables to accomplish the rules.

```
CREATE TRIGGER sal_constr
ON salary
FOR INSERT, UPDATE
AS
DECLARE @empid NUMERIC(10, 0),
        @salary_level NUMERIC(15, 4),
        @manager NUMERIC(10, 0),
        @manager_salary NUMERIC(15, 4),
        @dept_maxsql NUMERIC(15, 4),
        @deptid NUMERIC(5, 0)

DECLARE ins_curs CURSOR
FOR
SELECT empid, salary_level
FROM inserted
```


```
OPEN ins_curs
FETCH ins_curs INTO @empid, @salary_level

WHILE (@@sqlstatus!= 2)
BEGIN
    /* Get the manager and department of the employee */
    SELECT @manager, @deptid
    FROM employees
    WHERE empid = @empid

    /* Get the manager's salary */
    SELECT @manager_salary = salary_level
    FROM salary
    WHERE empid = @manager

    /* Get the maximum salary for the department */
    SELECT @dept_maxsal = max_salary
    FROM depts
    WHERE deptid = @deptid
```

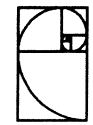



```
    /* Business Rule 1 */
    IF (@salary_level > @manager_salary)
        ROLLBACK TRANSACTION

    /* Business Rule 2 */
    IF (@salary_level > @dept_maxsal)
        ROLLBACK TRANSACTION

    FETCH ins_curs
END

CLOSE ins_curs
RETURN
```

Database Cursors

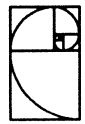
- A new stored procedure **sp_cursorinfo** will give information on the cursors declared in a session. This procedure will give information on various properties of the cursor, the memory allocated to the cursor, the target column list and some other useful information.
- **sp_lock** has been enhanced to display the locks in use by a cursor. It will print out an extra column to distinguish between cursor and non-cursor locks.



Database Cursors

- Example of sp_cursorinfo

```
> sp_cursorinfo
Cursor name 'c1' is declared at nesting level '0'.
The cursor id is 65539.
The cursor is positioned on a row.
There have been 1 rows read through this cursor.
The cursor will remain open when a transaction is committed
or rolled back.
The number of rows returned for each FETCH is 1.
The cursor is updatable.
This cursor is using 5132 bytes of memory.
There are 2 columns returned by this cursor.
The result columns are:
Name = 'name', Table = 'Table1', Type = VARCHAR,
Length = 30 (updatable)
Name = 'type', Table = 'Table1', Type = SMALLINT,
Length = 2
```

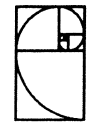



Database Cursors

- Example of sp_lock:

  >sp_lock
  The class column will display the cursor name for locks associated with a cursor for the current user and the cursor id for other users.

spid	locktype	table_id	page	dbname	class
1	Sh_intent	30	0	master	c1
1	Sh_page	30	257	master	c1
1	Sh_intent	384004399	0	master	Non Cursor Lock



Database Cursors

- A design for applications using cursors should consider the following factors:

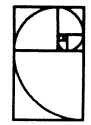
  1. How long will my cursor remain open?
  2. How many cursors will be open simultaneously?

     Answers to 1 & 2 will tell you the resource usage characteristics of your application.
  3. Will I be doing updates to the scanned tables in the same process when the cursor is open?

     If this is the case, then the cursor scan will be affected in much the same way as when these updates were done through the cursor.
  4. What is my criterion for uniquely identifying rows in the tables?

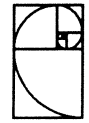
     The answer here could be a primary key which does not change or a column with the new Identity property.
  5. Is the cursor really necessary here?

     Once cursors are available, they can be abused. Remember the Relational Model is essentially a Set Oriented Model.



Database Cursors

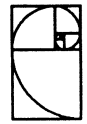
- Cursors are not needed when:
  1. The rows to be selected have the same uniform qualification criteria.
  2. The same operation is done to every row.
- Example: Give a bonus to all accounts which have a balance more than $2,000. Here a single update is much better than using a cursor to decide which rows to update.



Database Cursors

- The example about reconciling of accounts also contained an interesting problem. The use of @do_update variable could have been avoided, had I decided to use the following SELECT which restricted the rows sent by the server.

```
SELECT balance
FROM accounts
FOR UPDATE OF balance
WHERE balance < 500 OR balance >=1000
```

System 10

Migration of Applications and Data to Sybase From Oracle, DB2 and Non-Relational Systems

Sybase European Users' Conference

October 12-14, 1993

The Hague, Netherlands

By Jeff Richey

Migration Issues

Relational

Non-Relational

Application

Data

Transaction Model

Lock Modes

Run-Time

Application Migration

Did Application use standard SQL (ISO, ANSI, X/OPEN)

Modify ESQL statements

SQL data types
data type conversions from SQL to host language
host language support
stored procedures, DBRM

Precompile

command line options
levels of syntax and semantic checking
listing, target, isql files

Compile, Link

SQL run-time library

run against sample test data

Data Migration

Determine Strategies

Determine which Sybase Product

OmniSQL
Gateways

Create Data Definitions

tables
databses
procedures
indexes
...

Migrate the Data

sample data

Simulate thru tests

debug application
check data definitions
review transaction/locking model

Run simultaneously with current system

Transaction Model

Differences in model between Sybase, Oracle and DB2

isolation levels
Ansi/ISO model
Oracle model
DB2 model
Sybase model

Statements to Define Boundries

implicit beginning of transaction
explicit beginning of transaction
commit
rollback

Using Savepoints

Effects on Cursors

Simulate Application with test data

Lock Modes

Tablespace, Table

Page level

Row Level

Early vs Late locking

Non-Relational Data

Application

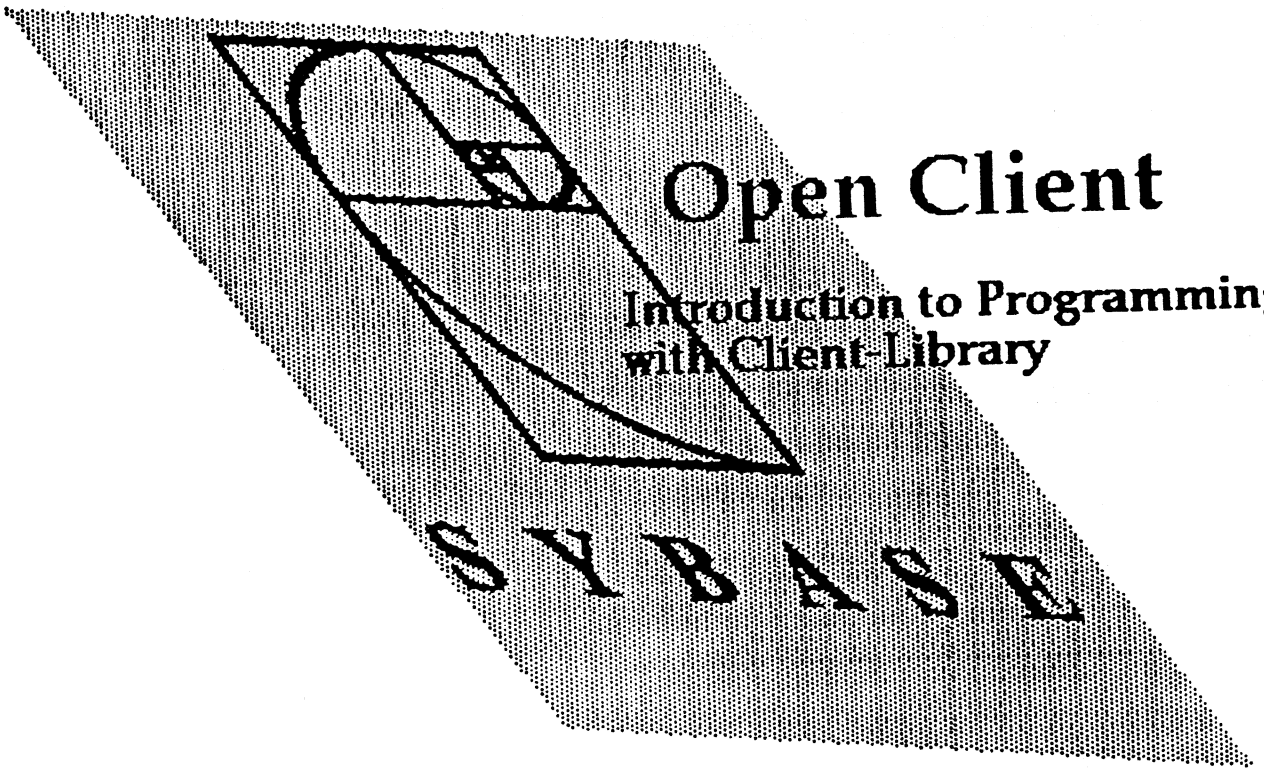
create ESQL file(s), use Modular Programming Feature
separate SQL from Application
create application to read/write data

Data

map data definitions to tables
normalize the data
create tables, indexes, databases, etc.

Transaction/Locking strategy

define boundries of transactions
determine locking modes for data



Open Client

Introduction to Programming with Client-Library

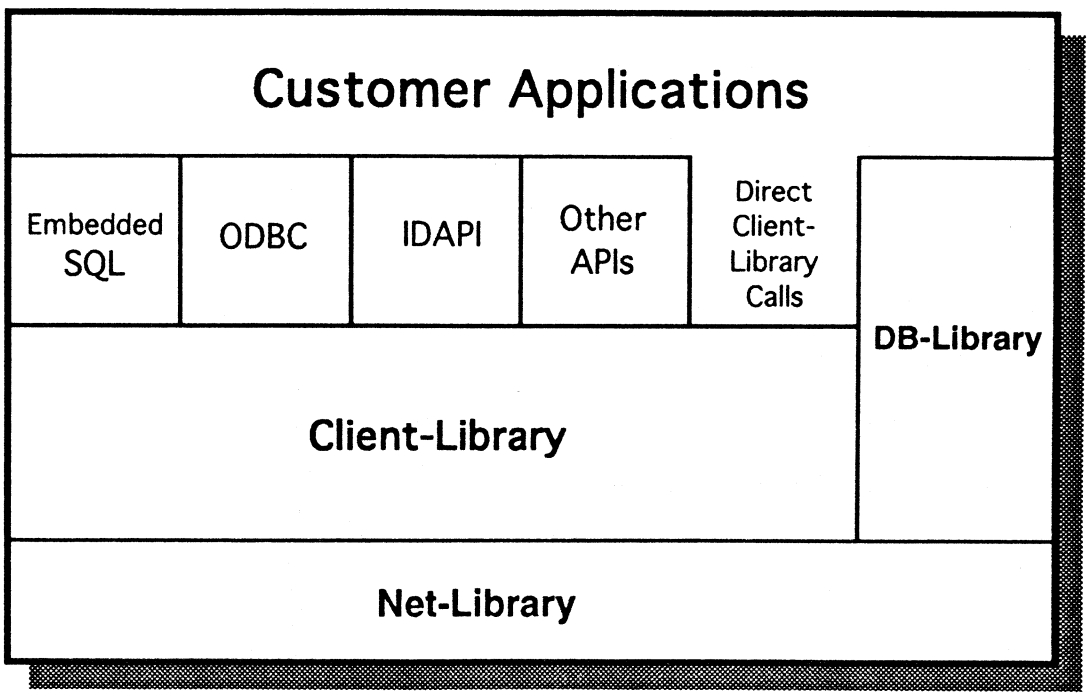
By Otto Lind

What Was Needed (Requirements)?

- Runtime support for PreCompilers
  - Native Cursors (Server side)
- Extensible to Support New and Future Server Features
- Runtime support for Standards
  - ODBC
  - IDAPI
- Support Future Communication Protocols
  - RDA
  - DRDA
- Take Advantage of Net-Library as the Transport Layer
- Maintain High Level of Performance
- Asynchronous I/O

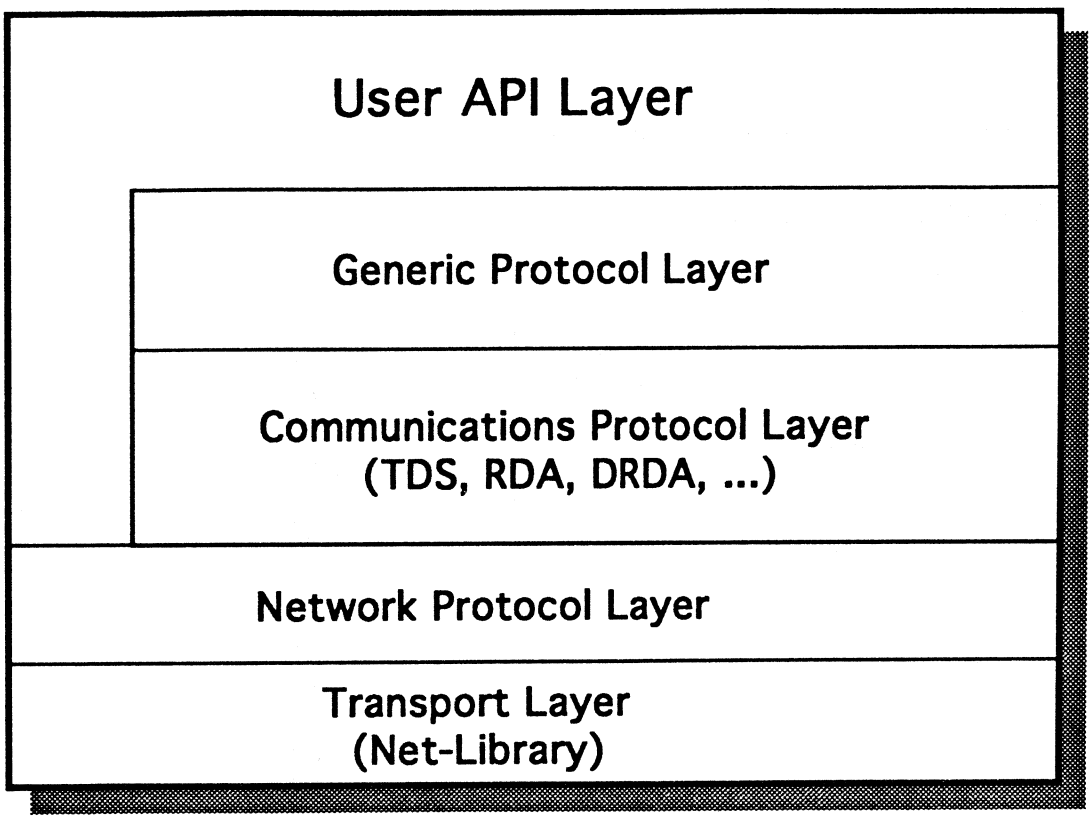
How Does Client-Library Support The Requirements?

- Supports all new System 10 Server features
  - Native Cursors
- Runtime Library
  - Layer beneath other development toolkits (ODBC, ESQL) as well as customer applications



How Does Client-Library Support The Requirements? (cont.)

- Client-Library itself is layered
  - Layered over Net-Library
  - Communication protocols (TDS, RDA, DRDA) are handled in a protocol layer

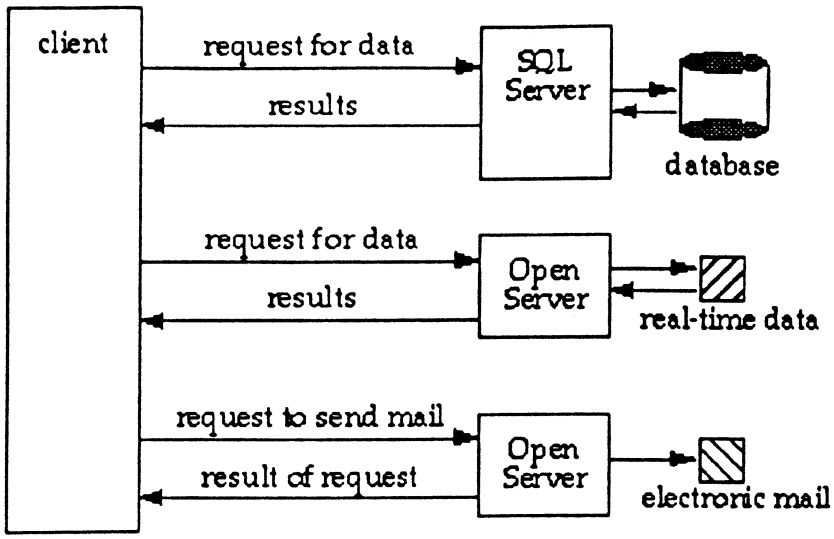


How Does Client-Library Support The Requirements? (cont.)

- Fully asynchronous I/O
- Provides performance features
  - Caches memory
  - Does not require building command lists (no dbcmd)
  - Does not require intermediate storage (no dbdata)
- Well integrated with OpenServer
  - Integrated thread support for connections
- System 10 DB-Library and Client-Library calls can be intermixed in an application

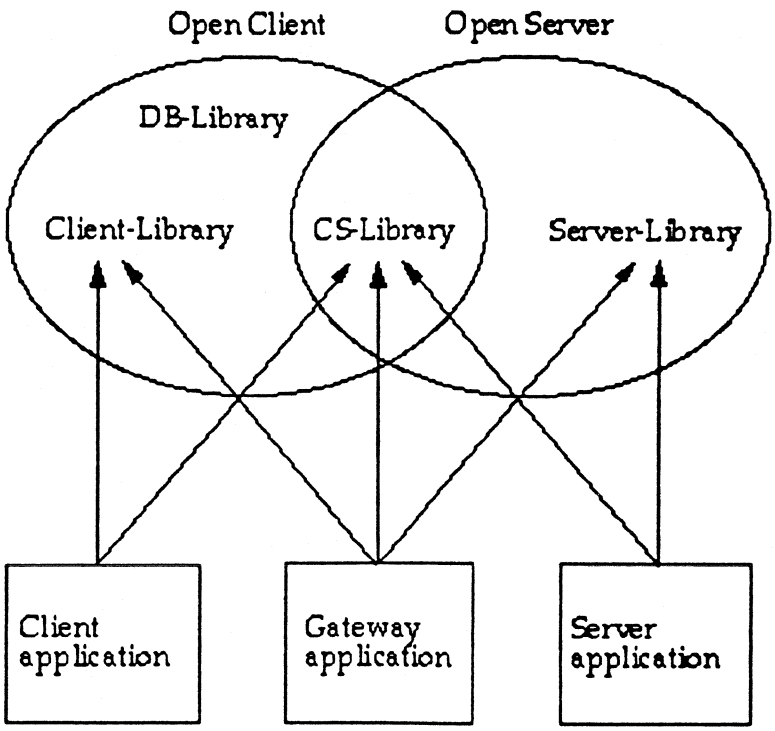
What About DB-Library

- It will continue to exist and be supported
- System 10 DB-Library can coexist with Client-Library in applications



Examples of Client/Server Applications using SQLServer and Open Server

• Open Client provides the programming interfaces and network services for use in writing client applications.



Applications using Open Client Libraries

• Open Client is composed, in part, by three libraries: Client-Library, DB-Library, and CS-Library. These three libraries make up the Open Client programming interface.

• Client-Library is a collection of routines for use in writing client applications. It is designed to accommodate the advanced features of the Sybase 10.0 product line, and asynchronous IO.

• CS-Library is a collection of utility routines for use in application programs.

• All CS-Library routines are available to both Open Client applications and Open Server applications, but a couple CS-Library routines are required in all Client-Library applications, cs_ctx_alloc and cs_ctx_drop.

• A Client-Library program is compiled and run in the same way as any other C language program.

Examples of Exposed Client-Library Structures

CS_DATAFMT Data Structure

```
typedef struct _cs_datafmt
{
 CS_CHAR name[CS_MAX_NAME]; /* Name of data */
 CS_INT  namelen;     /* Length of name */
 CS_INT  datatype;    /* Datatype of data */
 CS_INT  format;      /* Format symbols */
 CS_INT  maxlength;   /* Max length of data */
 CS_INT  scale;       /* Scale of data */
 CS_INT  precision;   /* Precision of data */
 CS_INT  status;      /* Status symbols */
 CS_INT  count;       /* number of rows to copy,
                         per ct_fetch call */
 CS_INT  usertype;    /* Svr user-def'd type */
 CS_LOCALE  *locale; /* Locale information */

} CS_DATAFMT;
```

CS_IODESC Data Structure

```
typedef struct _cs_iodesc
{
 CS_INT     iotype;          /* CS_IODATA*/
 CS_INT     datatype;        /* Text or image */
 CS_LOCALE *locale;          /* Locale information */
 CS_INT       usertype;      /* User-defined type */
 CS_INT         total_txtlen; /* Total data length */
 CS_INT           offset;     /* Reserved */
 CS_BOOL      log_on_update;  /* Log the update */
 CS_CHAR      name[CS_OBJ_NAME]; /* Name of data
                                    object */
 CS_INT     namelen ;  /* Length of name */
 CS_BYTE  timestamp[CS_TS_SIZE]; /* SQL Server id */
 CS_INT   timestamplen ; /* Length of timestamp */
 CS_BYTE  textptr[CS_TP_SIZE]; /* SQL Server ptr */
 CS_INT     textptrlen ;  /* Length of textptr */

} CS_IODESC;
```

- A **CS_DATAFMT** structure is used to describe data values and program variables.
- A **CS_IODESC** structure is used to describe text or image data.

Examples of Exposed Client-Library Structures

CS_CLIENTMSG Data Structure

```
typedef struct _cs_clientmsg
{
 CS_INT          severity;
 CS_MSGNUM       msgnumber;
 CS_CHAR         msgstring[CS_MAX_MSG];
 CS_INT          msgstringlen;
  /*
 ** Operating-system-specific information:
  */
 CS_INT          osnumber;
 CS_CHAR         osstring[CS_MAX_MSG];
 CS_INT          osstringlen;

} CS_CLIENTMSG;
```

+ sql state
+ transaction state
(rolled back
committed
etc.)
+ severity

CS_SERVERMSG Data Structure

```
typedef struct _cs_servermsg
{
 CS_MSGNUM       msgnumber;
 CS_INT          state;
 CS_INT          severity;
 CS_CHAR         text[CS_MAX_MSG];
 CS_INT          textlen;
 CS_CHAR         srvname[CS_MAX_NAME];
 CS_INT          srvnlen;
  /*
 ** Stored procedure information
  */
 CS_CHAR         proc[CS_MAX_NAME];
 CS_INT          proclen;
 CS_INT          line;
  /*
 ** Other information.
  */
 CS_INT          status;
} CS_SERVERMSG;
```

- A **CS_CLIENTMSG** structure is used to describe a Client-Library error or informational message.

- A **CS_SERVERMSG** structure is used to describe a server error or informational message.

- These structures are used by an application during error handling.

Calling Convention for the Client Library API

CS_RETCODE ct_xxxx(handle, input parameters, output parameters)

- All Client-Library calls return a type CS_RETCODE which represents the overall state of the function's activity. Typical return codes are CS_SUCCEED, CS_BUSY, CS_PENDING, CS_FAIL.

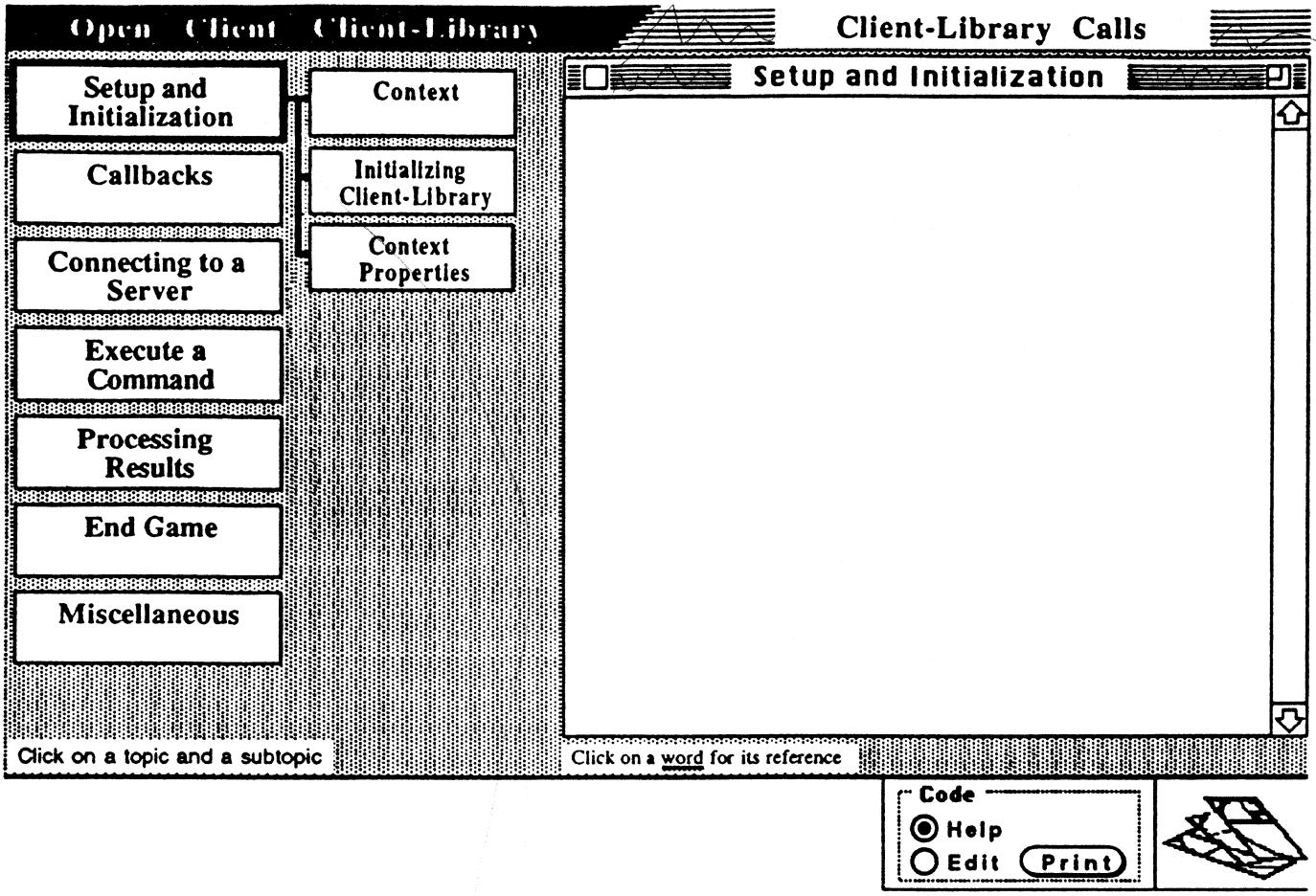
- Specific error conditions are handled through callback message handlers, or inline error handling, using **ct_diag.**

- All Client-Library routines start with the prefix **ct_**.

- The first parameters passed to all Client-Library routines are pointers to hidden structures, refered to as **handles**.

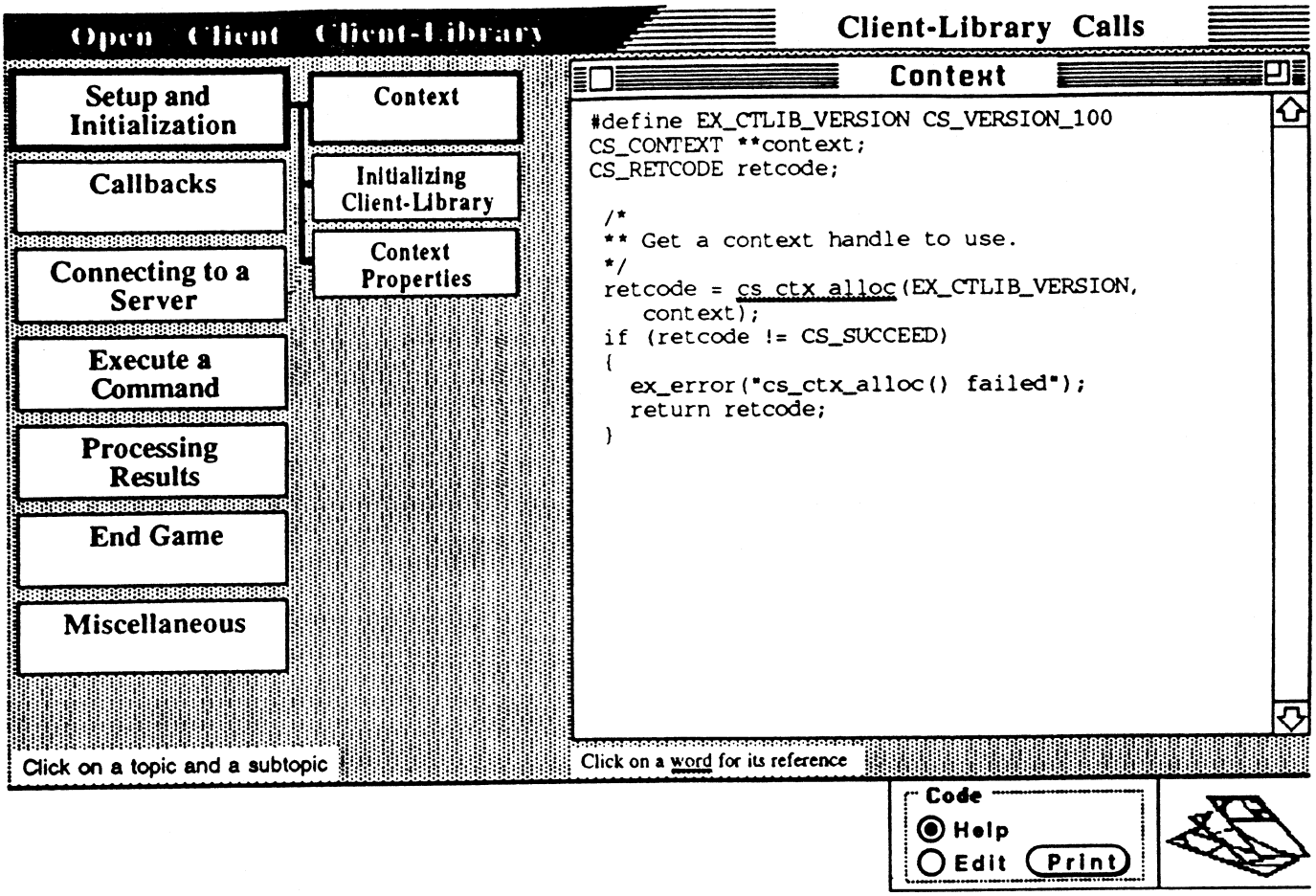
- After the handle(s), **input parameters** are passed.

- After the input parameter(s), **output parameters** are passed.



• A Client-Library program must first configure the environment, or context, within which Client-Library will operate.

• Second, it must initialize Client-Library, so Client-Library can setup its internal control structures and define the version behavior the application will expect.



- A Context defines an operating environment for a set of server connections. An application can have more than one context, however a simple application typically has only one.

- A CS_CONTEXT structure is used to store the context information.

- A context structure must be allocated, using cs_ctx_alloc, before initializing Client-Library.

cs_ctx_alloc

Function
Allocate a CS_CONTEXT structure.

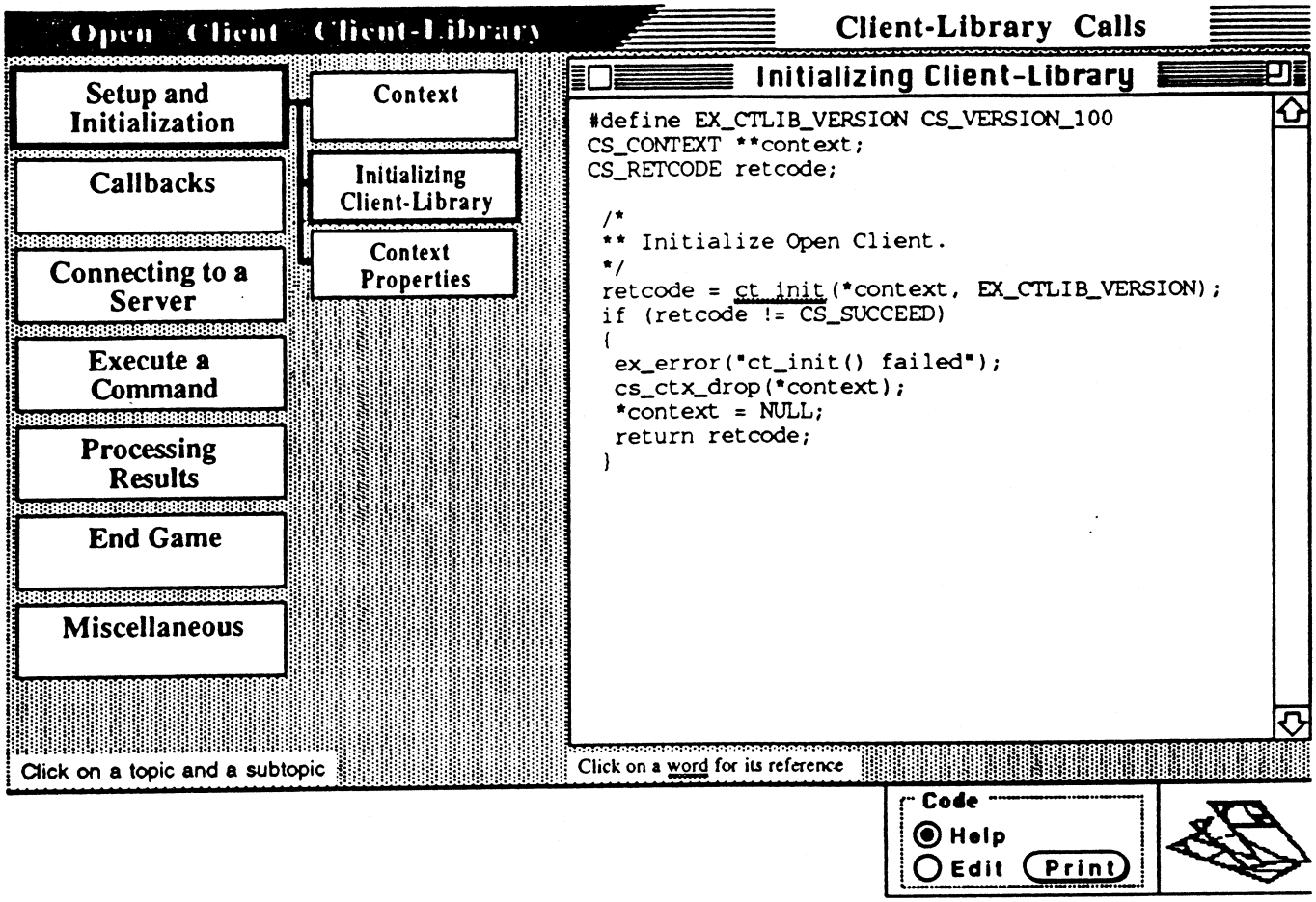
Syntax
```
CS_RETCODE cs_ctx_alloc(version, ctx_pointer)

CS_INT version;
CS_CONTEXT **ctx_pointer;
```

Parameters
version - The version of CS-Library behavior that the application expects.
ctx_pointer - The address of a pointer variable. cs_ctx_alloc sets *ctx_pointer to the address of a newly-allocated CS_CONTEXT structure.
In case of error, cs_ctx_alloc sets *ctx_pointer to NULL.

Comments
- cs_ctx_alloc allocates a CS_CONTEXT structure.
- A CS_CONTEXT structure, also called a "context structure," contains information that describes an application context. For example, a context structure contains default localization information, and defines the version of CS-Library that is in use.
- Allocating a context structure is the first step in any Client-Library or Server-Library application.
- After allocating a CS_CONTEXT, a Client-Library application typically customizes the context by calling cs_config and/or ct_config, and then sets up one or more connections within the context. A Server-Library application can customize a context by calling cs_config and srv_props.
- To de-allocate a context structure, an application can call cs_ctx_drop.
- cs_ctx_global also allocates a context structure. The difference between cs_ctx_alloc and cs_ctx_global is that cs_ctx_alloc allocates a new context structure each time it is called, while cs_ctx_global allocates a new context structure only once, the first time it is called. On subsequent calls, cs_ctx_global simply returns a pointer to the existing context structure.

Returns
CS_SUCCEED - The routine completed successfully.
CS_MEM_ERROR - The routine failed because it could not allocate sufficient memory.
CS_FAIL - The routine failed for other reasons.

See Also
ct_con_alloc, ct_config, cs_ctx_drop, cs_ctx_global, cs_config



- ct_init sets up internal control structures and defines the version of Client-Library behavior that the application requires.

- ct_init must be the first Client-Library call in an application.

ct_init

Function
Initialize Client-Library.

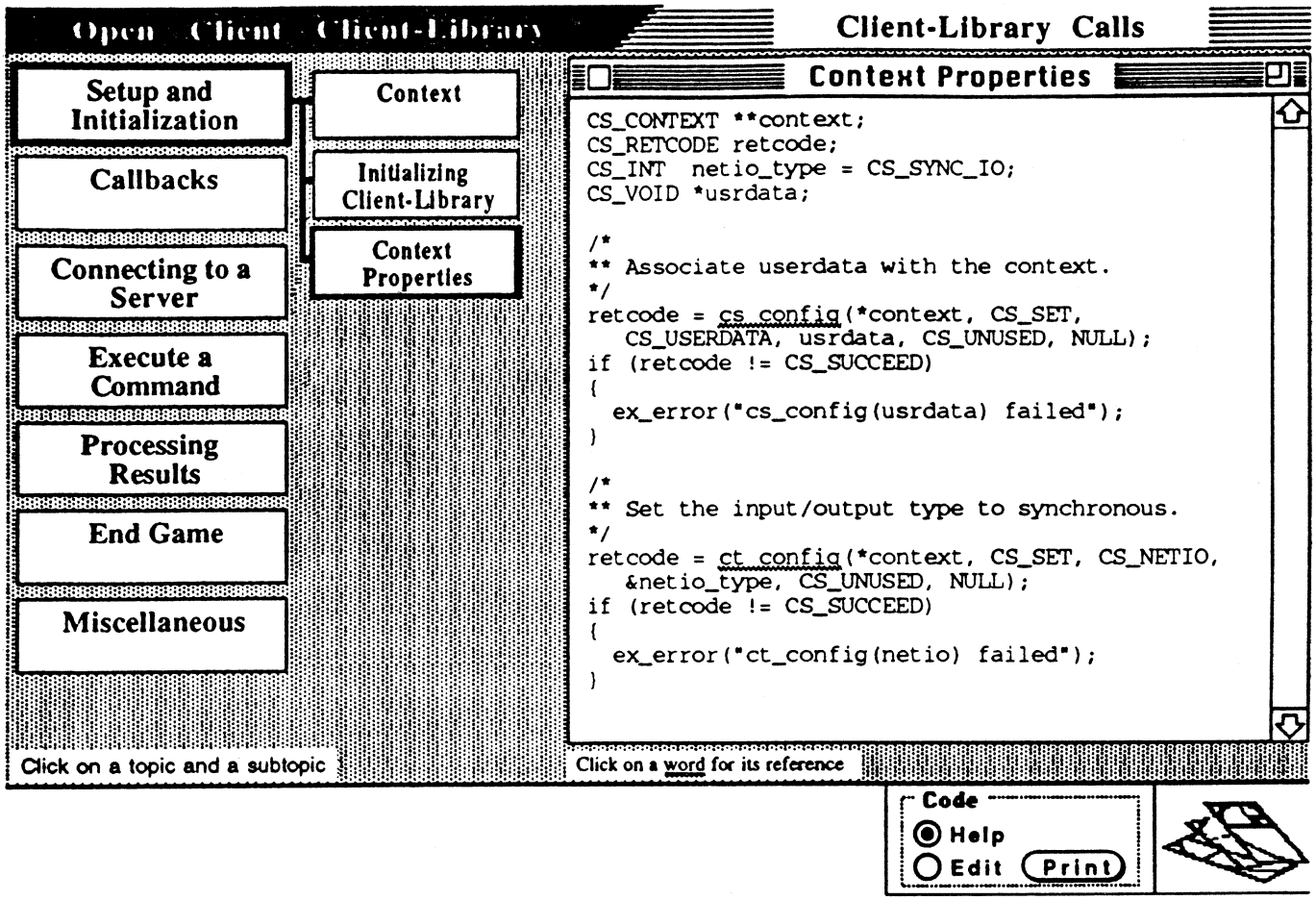
Syntax
```
CS_RETCODE ct_init(context, version)

CS_CONTEXT *context;
CS_INT version;
```

Parameters
context - A pointer to a CS_CONTEXT structure. An application must have previously allocated this context structure by calling the CS-Library routine cs_ctx_alloc. context identifies the Client-Library context being initialized.
version - The version of Client-Library behavior that the application expects.

Comments
- ct_init initializes Client-Library for an application context. It sets up internal control structures and defines the version of Client-Library behavior that the application expects.
- If ct_init returns CS_SUCCEED, Client-Library will provide the requested behavior, regardless of the actual version of Client-Library in use. If Client-Library cannot provide the requested behavior, ct_init returns CS_FAIL. Generally speaking, higher-level versions of Client-Library can provide lower-level behavior, but lower versions cannot provide higher-level behavior.
- ct_init must be the first Client-Library routine called in a Client-Library application context. Other Client-Library routines will fail if they are called before ct_init. A Client-Library application can call CS-Library routines before calling ct_init (and in fact must call the CS-Library routine cs_ctx_alloc before calling ct_init).
- Because an application calls ct_init before it sets up error handling, an application must check ct_init's return code to detect failure.
- It is not an error for an application to call ct_init multiple times. Some applications cannot guarantee which of several modules will execute first. In such a case, each module should contain a call to ct_init.
- version is the version of Client-Library behavior that the application expects. version determines the value of the context's CS_VERSION property. Connections allocated within a context use default CS_TDS_VERSION values based on their parent context's CS_VERSION level.

Returns
CS_SUCCEED - The routine completed successfully.
CS_MEM_ERROR - The routine failed due to a memory allocation error.
CS_FAIL - The routine failed for other reasons. ct_init returns CS_FAIL if Client-Library cannot provide version-level behavior.

See Also
cs_ctx_alloc, ct_exit, ct_keydata



- Contexts properties define a context's behavior.

- Some properties apply directly to the context while others serve to define default behavior for connections created within the context.

- There are CS-Library properties and Client-Library properties that can be defined for the context.

- Some properties can be changed at any time during the application, effecting future behavior.

- Examples of properties are:
- CS_LOGIN_TIMEOUT : The login timeout value.
- CS_VERSION : The version of Client-Library in use by this context.
- CS_NETIO : Asynchronous or synchronous I/O.
- CS_NOINTERRUPT : Whether or not the application can be interrupted.

cs_config

Function
Set or retrieve CS-Library properties.

Syntax
CS_RETCODE cs_config(context, action, property, buffer, buflen, outlen)

```
CS_CONTEXT *context;
CS_INT action;
CS_INT property;
CS_VOID *buffer;
CS_INT buflen;
CS_INT *outlen;
```

Parameters
context - A pointer to a CS_CONTEXT structure.
action - An integer variable which has been set to one of the following symbolic values:
CS_SET : Sets the value of the property.
CS_GET : Retrieves the value of the property.
CS_CLEAR : Clears the value of the property by resetting it to its default value.
property - The property whose value is being set or retrieved. The following lists the symbolic values that are legal for property:
CS_EXTRA_INF : Whether or not to return the extra information that's required when processing messages in-line, using a SQLCA or SQLCODE.
CS_LOC_PROP : A CS_LOCALE structure that defines localization information for this context.
CS_MESSAGE_CB : The CS-Library message handler callback.
CS_USERDATA : User-allocated data.
CS_VERSION : The version of CS-Library.
buffer - If a property value is being set, buffer points to the value to use in setting the property. If a property value is being retrieved, buffer points to the space in which cs_config will place the value of the property. If a property value is being cleared, pass buffer as CS_UNUSED.
buflen - The length, in bytes, of *buffer. If a property value is being set and the value in *buffer is null-terminated, pass buflen as CS_NULLTERM. If a property value is being set and *buffer is a fixed-length or symbolic value, pass buflen as CS_UNUSED. If a property value is being retrieved and buflen indicates that *buffer is not large enough to hold the requested information, cs_config sets *outlen to the length of the requested information and returns CS_FAIL.
outlen - A pointer to an integer variable. outlen is not used if a property value is being set. If a property value is being retrieved, cs_config sets *outlen to the length, in bytes, of the requested information. If the information is larger than buflen bytes, an application can use the value of *outlen to determine how many bytes are needed to hold the information. If an application is setting a property value or does not care about return length information, it can pass outlen as NULL.

Comments
• cs_config sets or retrieves the values of CS-Library context properties.
• There are three kinds of context properties:
- Context properties specific to CS-Library.
- Context properties specific to Client-Library.
- Context properties specific to Server-Library.
cs_config sets and retrieves the values of CS-Library context properties. With the exception of CS_LOC_PROP, properties set via cs_config affect only CS-Library.
ct_config sets and retrieves the values of Client-Library-specific context properties. Properties set via ct_config affect only Client-Library.
srv_props sets and retrieves the values of Server-Library-specific context properties. Properties set via srv_props affect only Server-Library.
• See the Properties topics page in the Client-Library Reference Manual for information on Client-Library properties.
About CS-Library Properties
• Extra Information
- CS_EXTRA_INF determines whether or not CS-Library returns the extra information that is required to fill in a SQLCA or SQLCODE structure.
- If an application is not retrieving messages into a SQLCA or SQLCODE, the extra information is returned as ordinary CS-Library messages.
• Locale Information

- The CS_LOC_PROP property defines a CS_LOCALE structure that contains localization information for a context. Localization information includes a language, character set, datetime, money, and numeric formats, and a collating sequence.
- CS_LOC_PROP affects both CS-Library and Client-Library, because a new connection picks up default localization information from its parent context.
- If an application does not call cs_config to define localization information for a context, the context uses default localization information that it picks up from the operating system environment when it is allocated. If localization information is not available in the operating system environment, the context uses platform-specific default localization values.
- An application can call cs_loc_alloc to allocate a CS_LOCALE structure.
• CS-Library Message Callback
- The CS_MESSAGE_CB property defines a pointer to a CS-Library message callback routine.
- An application installs Client-Library callback routines by calling ct_callback. An application installs a CS-Library message callback routine by calling cs_config to set the value of the CS_MESSAGE_CB property. Aside from this difference, the CS-Library message callback is very similar to the Client-Library client message callback.
- For specific information on how to define a CS-Library message callback routine, see Chapter 1, "Introduction to CS-Library." For general information on callback routines, see the Callbacks topics page in the Client-library Reference Manual.
• User-Allocated Data
- The CS_USERDATA property defines user-allocated data. This property allows an application to associate user data with a particular context structure.
- CS-Library copies the user data into internal data space. An application can then call cs_config at a later time to retrieve the data.
- Although Client-Library also has a CS_USERDATA property, the Client-Library CS_USERDATA is set only at the connection and command levels.
• Version Level
- The CS_VERSION property represents the version of CS-Library behavior that an application has requested via cs_ctx_alloc.
- An application can only retrieve the value of CS_VERSION.
- Currently, the only value that is legal for CS_VERSION is CS_VERSION_100.

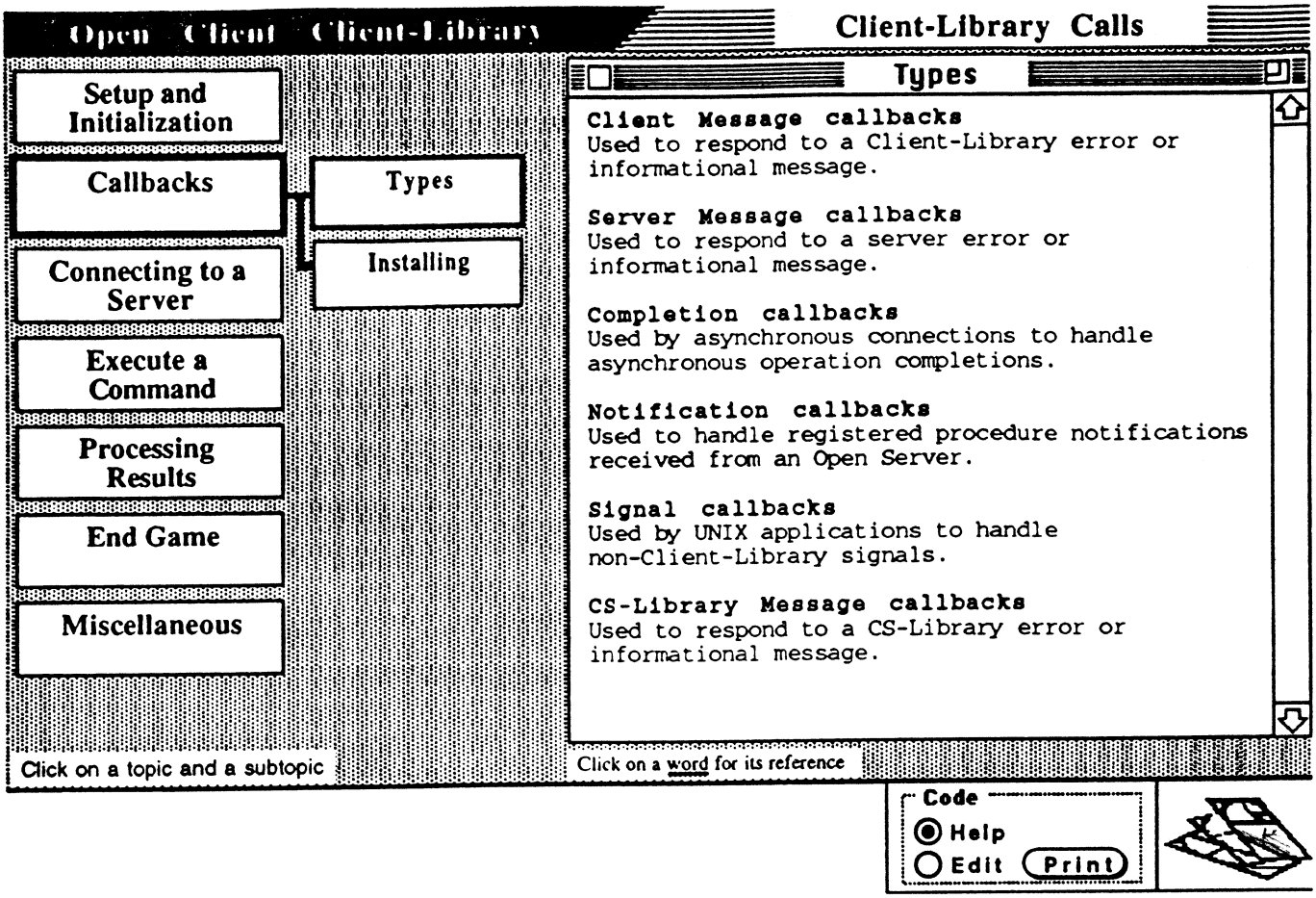
Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.

See Also
cs_ctx_alloc, ct_con_props, ct_config, ct_init

CS_TEXTLIMIT : The largest text or image value to be returned on this connection.
CS_TIMEOUT : The timeout value.
CS_VERSION : The version of OC-Library in use by this context.

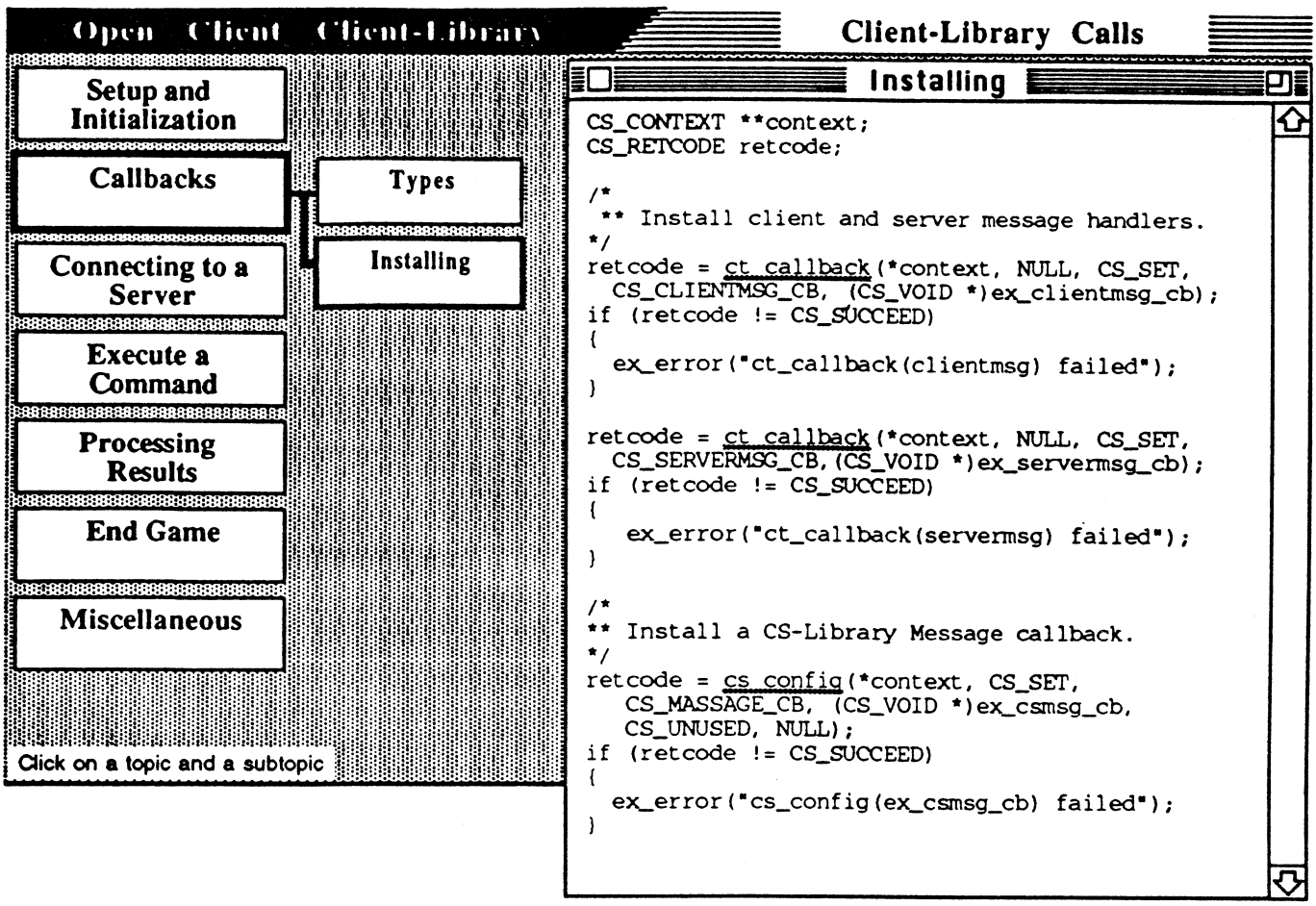
Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.

See Also
cs_config, ct_cmd_props, ct_capability, ct_con_props, ct_connect, ct_init, Properties
ct_connect



• Callbacks are most commonly used to handle Client-Library and server error and informational messages.

• Inline error handling can also be used, via **ct_diag**, as a substitute to callback error handling.



- An application installs a callback routine by passing a pointer to the callback routine and the callback type to the routine ct_callback.

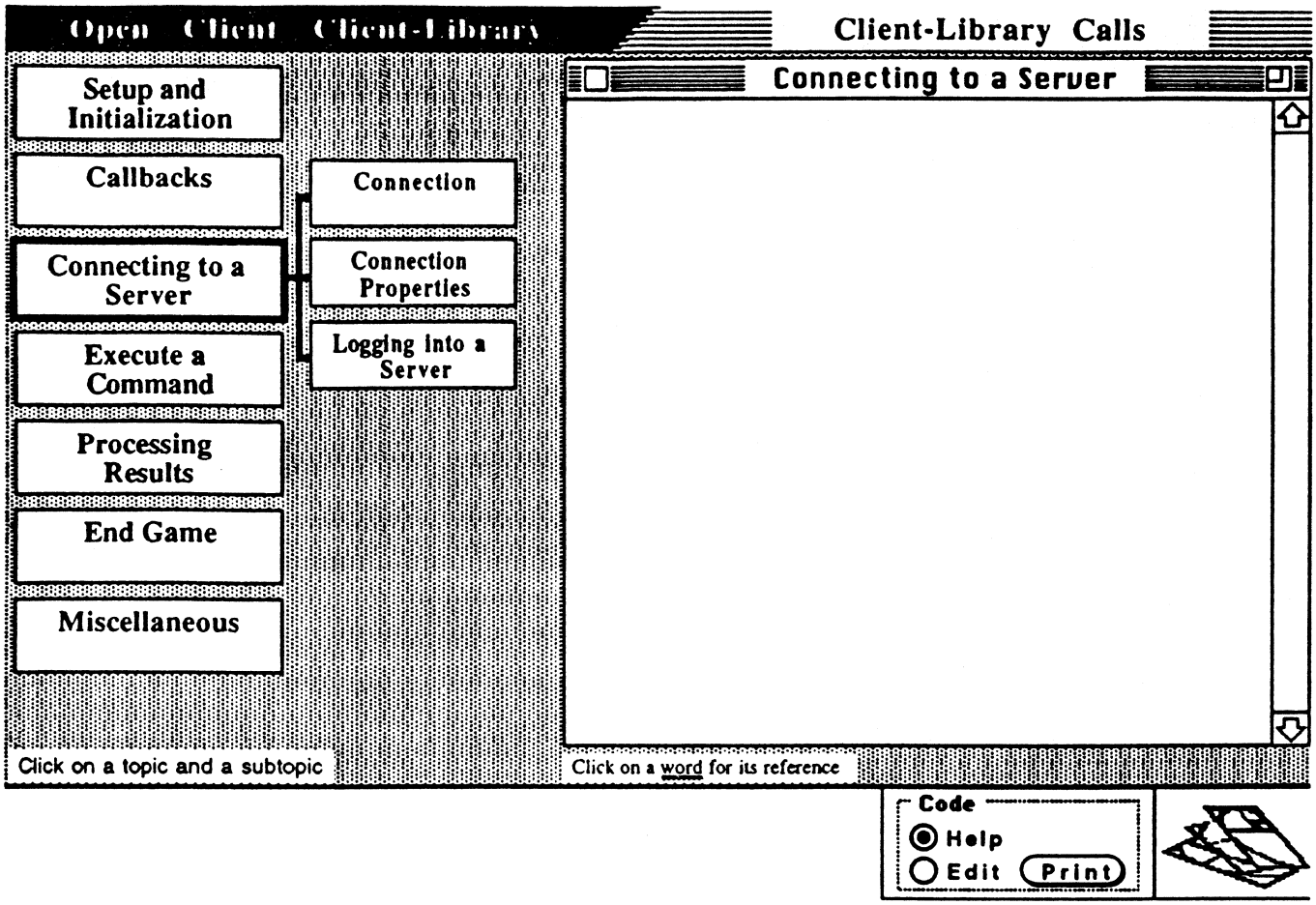
- Callbacks can be installed at the context or connection level.

- ct_callback can also be used to de-install or replace a callback routine, and retrieve a pointer to the current callback routine for a particular type.

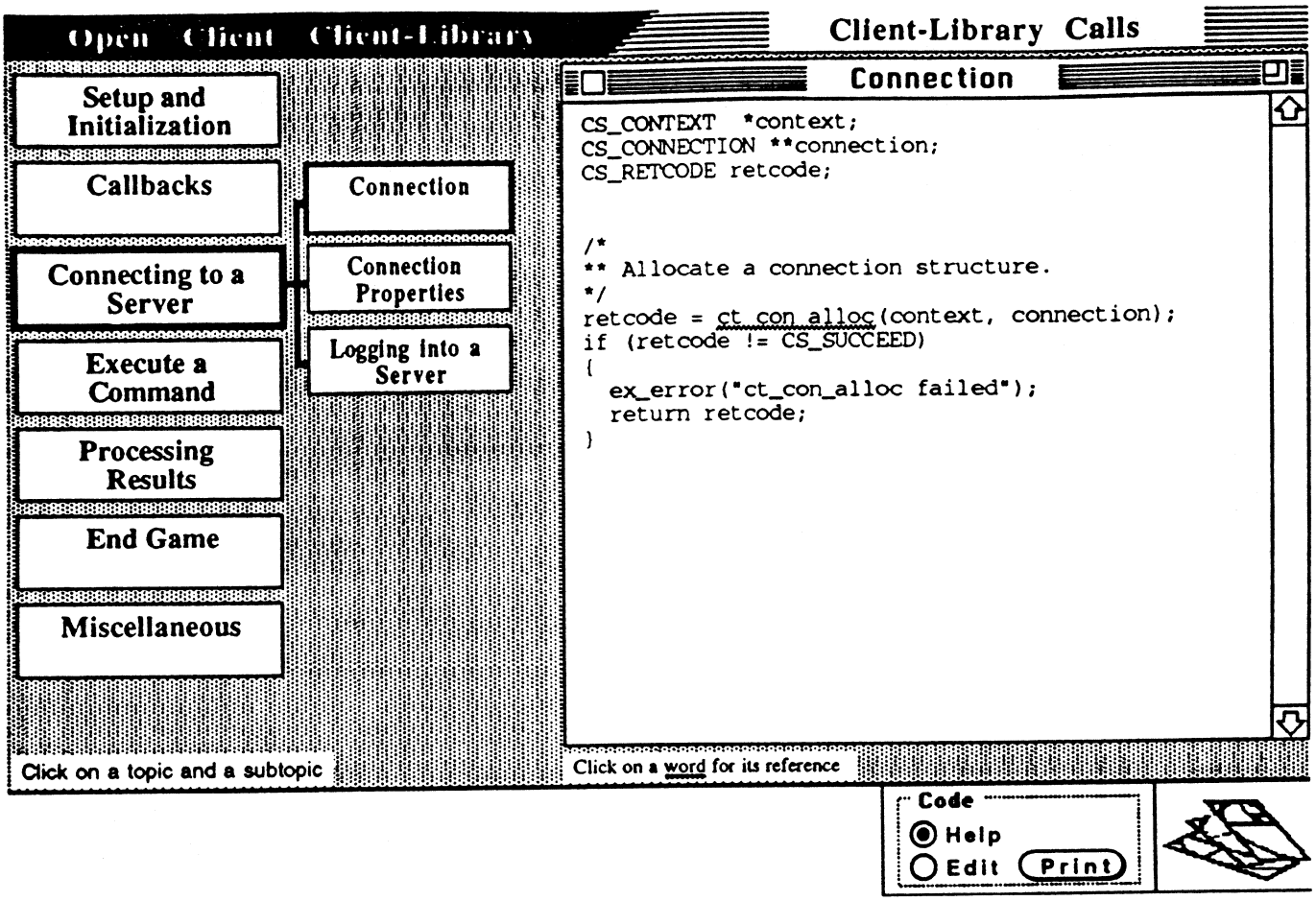
topics page.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
Callbacks, ct_capability, ct_config, ct_con_props, ct_connect, ct_cancel



• Three step process; Allocate a connection structure, set it properties, and log into the server.



- An application calls **ct_con_alloc** to allocate a connection structure.

ct_con_alloc

Function
Allocate a CS_CONNECTION structure.

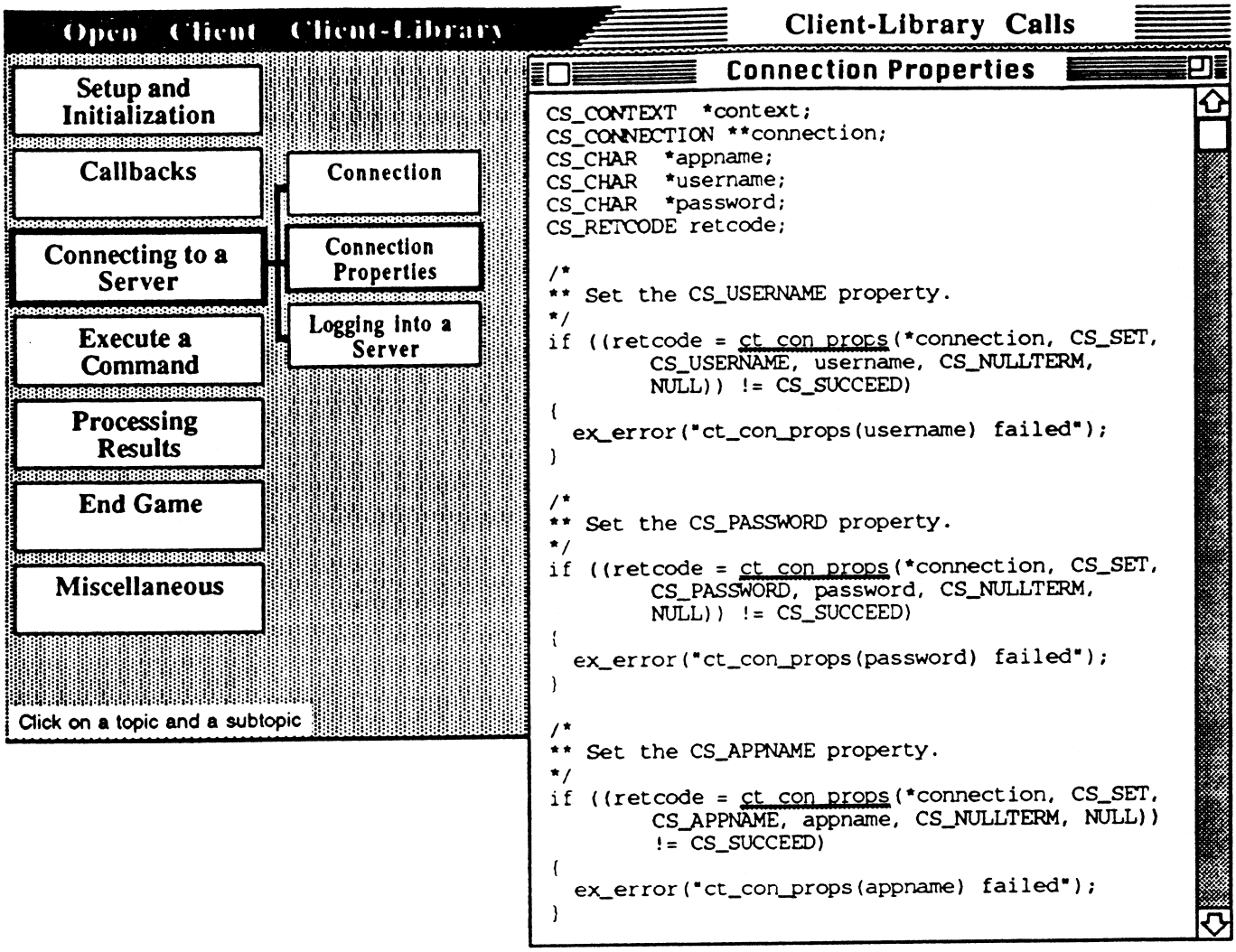
Syntax
```
CS_RETCODE ct_con_alloc(context, con_pointer)

CS_CONTEXT *context;
CS_CONNECTION **con_pointer;
```

Parameters
context - A pointer to a CS_CONTEXT structure.
con_pointer - The address of a pointer variable. ct_con_alloc sets *con_pointer to the address of a newly-allocated CS_CONNECTION structure. In case of error, ct_con_alloc sets *con_pointer to NULL.

Comments
- ct_con_alloc allocates a CS_CONNECTION structure.
- A CS_CONNECTION structure, also called a "connection structure," contains information about a particular client/server connection.
- Before calling ct_con_alloc, an application must allocate a context structure by calling the CS-Library routine cs_ctx_alloc, and must initialize Client-Library by calling ct_init.
- Connecting to a server is a three-step process. To connect to a server, an application:
  - Calls ct_con_alloc to allocate a CS_CONNECTION structure.
  - Calls ct_con_props to set the values of connection-specific properties, if desired.
  - Calls ct_connect to create the connection and log in to the server.
- An application can have multiple open connections to one or more servers at the same time. Each server connection requires a separate CS_CONNECTION structure.
- In order to send commands to a server, one or more command structures must be allocated for a connection. ct_cmd_alloc allocates a command structure.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.

See Also
cs_ctx_alloc, ct_cmd_alloc, ct_close, ct_connect, ct_con_props, ct_con_drop



• An application calls **ct_con_props** to set, retrieve, or clear connection structure properties.

• Connection properties define various aspects of a connection's behavior. For example, the CS_PASSWORD property defines the password that a connection will use when logging into a server.

• When a connection structure is allocated, it picks up default property values from its parent context. Other properties (those that don't exist at the context level, such as CS_PACKETSIZE) default to standard Client-Library values.

• Some examples of connection properties are:
- CS_TEXTLIMIT : The largest text or image value to be returned on the connection.
- CS_APPNAME : The application name when logging into the server.
- CS_PARENT_HANDLE : The address of the parent context structure.

ct_con_props

Function
Set or retrieve connection structure properties.

Syntax
CS_RETCODE ct_con_props(connection, action, property, buffer, buflen, outlen)

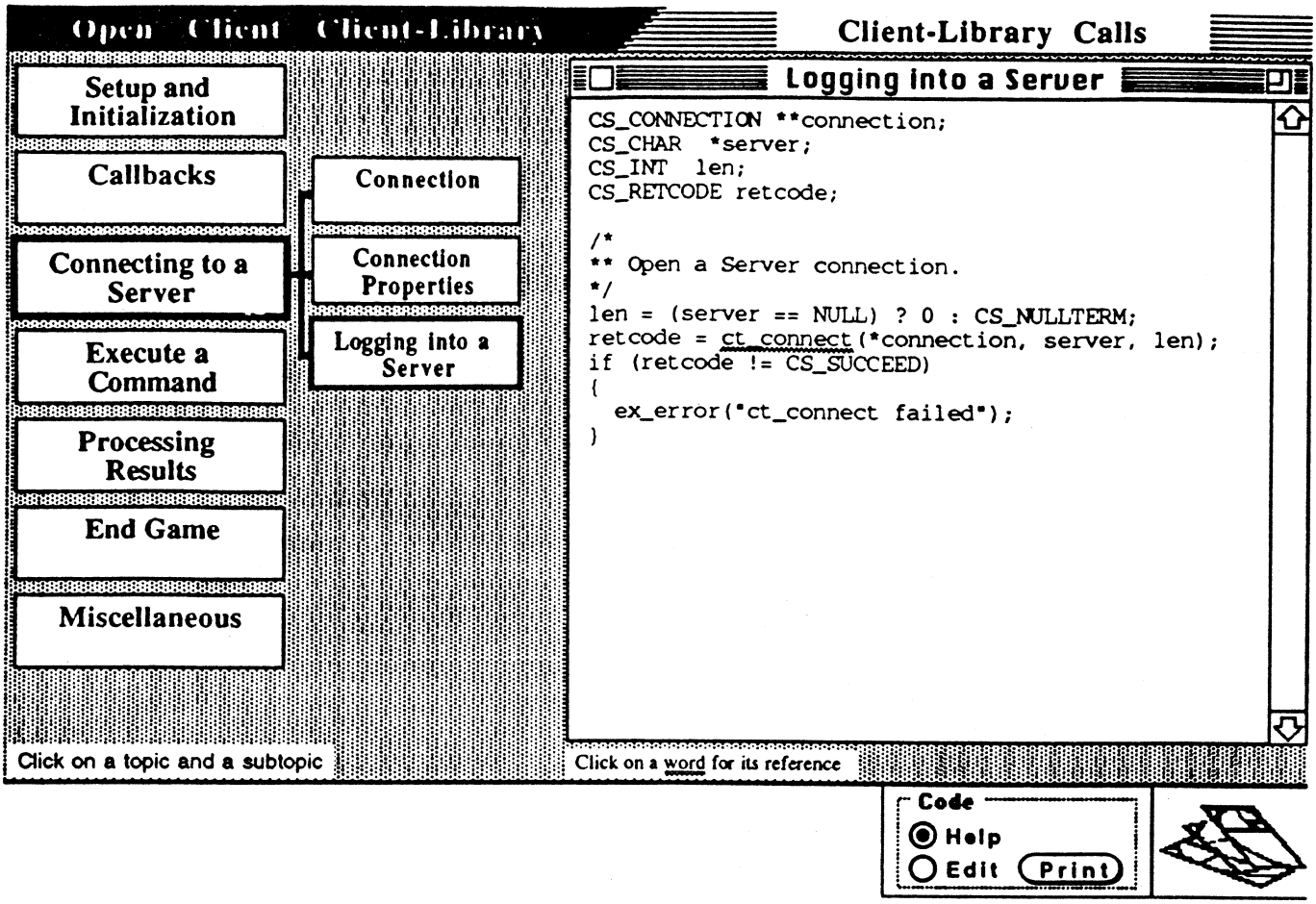
```
CS_CONNECTION *connection;
CS_INT action;
CS_INT property;
CS_VOID *buffer;
CS_INT buflen;
CS_INT *outlen;
```

Parameters
connection - A pointer to a CS_CONNECTION structure. A CS_CONNECTION structure contains information about a particular client/ server connection.
action - An integer variable which has been set to one of the following symbolic values:
CS_SET : Sets the value of the property.
CS_GET : Retrieves the value of the property.
CS_CLEAR : Clears the value of the property by resetting it to its Client-Library default value.
property - The symbolic name of the property whose value is being set or retrieved. The Properties topics page lists Client-Library properties.
buffer - If a property value is being set, buffer points to the value to use in setting the property. If a property value is being retrieved, buffer points to the space in which ct_con_props will place the requested information.
buflen - The length, in bytes, of *buffer. If a property value is being set and the value in *buffer is null-terminated, pass buflen as CS_NULLTERM.
outlen - A pointer to an integer variable. outlen is not used if a property value is being set and should be passed as NULL. If a property value is being retrieved and outlen is supplied, ct_con_props sets *outlen to the length, in bytes, of the requested information. If the information is larger than buflen bytes, an application can use the value of *outlen to determine how many bytes are needed to hold the information.
Comments
• ct_con_props sets or retrieves the values of connection structure properties.
• Connection properties define aspects of Client-Library behavior at the connection level.
• All command structures allocated for a connection pick up default property values from the parent connection. An application can override these default values by calling ct_cmd_props to set property values at the command structure level.
• If an application changes connection property values after allocating command structures for the connection, the existing command structures will not pick up the new property values. New command structures allocated for the connection will use the new property values as defaults.
• Some connection properties only take effect if they are be set before an application calls ct_connect to establish the connection. See the "Notes" column in Table 3-34: Client-Library Connection Properties, on the following page.
• See the Properties topics page for more information on properties.
• An application can use ct_con_props to set or retrieve the following properties:
CS_ANSI_BINDS : Whether or not to use ANSI-style binds.
CS_APPNAME : The application name used when logging into the server.
CS_BULK_LOGIN : Whether or not a connection is enabled to perform bulk copy "in" operations.
CS_CHARSETCNV : Whether or not character set conversion is taking place.
CS_COMMBLOCK : A pointer to a communication sessions block.
CS_DIAG_TIMEOUT_FAIL : When in-line error handling is in effect, whether Client-Library should fail or retry on timeout errors.
CS_EED_CMD : A pointer to a command structure containing extended error data.
CS_EXPOSE_FMTS : Whether or not to expose results of type CS_ROWFMT_RESULT and CS_COMPUTEFMT_RESULT.
CS_EXTRA_INF : Whether or not to return the extra information that's required when processing Client-Library messages in-line using a SQLCA or SQLCODE.
CS_HIDDEN_KEYS : Whether or not to expose hidden keys.
CS_HOSTNAME : The host machine name.
CS_LOC_PROP : A CS_LOCALE structure that defines localization information.
CS_LOGIN_STATUS : Whether or not the connection is open.

CS_NETIO : Whether network I/O is synchronous or asynchronous.
CS_NOINTERRUPT : Whether or not the application can be interrupted.
CS_NOTIF_CMD : A pointer to a command structure containing registered procedure notification parameters.
CS_PACKETSIZE : The TDS packet size.
CS_PARENT_HANDLE : The address of the connection structure's parent context.
CS_PASSWORD : The password used to log into the server.
CS_TDS_VERSION : The version of the TDS protocol that the connection is using.
CS_TEXTLIMIT : The largest text or image value to be returned on this connection.
CS_TRANSACTION_NAME : A transaction name.
CS_USERDATA : User-allocated data.
CS_USERNAME : The name used to log into the server.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
ct_capability, ct_cmd_props, ct_connect, ct_config, ct_init, Properties



• An application calls **ct_connect** to connect to a server. In the process of establishing a connection, **ct_connect** sets up communication with the network, logs into the server, and communicates any connection-specific property information to the server.

ct_connect

Function
Connect to a server.

Syntax
CS_RETCODE ct_connect(connection, servername, snamelen)

CS_CONNECTION *connection;
CS_CHAR *server_name;
CS_INT snamelen;

Parameters
connection - A pointer to a CS_CONNECTION structure. A CS_CONNECTION structure contains information about a particular client/ server connection.
servername - A pointer to the name of the server to connect to.
snamelen - The length, in bytes, of *server_name. If *server_name is null-terminated, pass snamelen as CS_NULLTERM.

Comments
• ct_connect establishes a connection between an application and a server. Information about the connection is stored in a CS_CONNECTION structure, which uniquely identifies the connection. In the process of establishing a connection, ct_connect sets up communication with the network, logs into the server, and communicates any connection-specific property information to the server.
• Because creating a connection involves logging into a server, an application must define login parameters (such as a server user name and password) before calling ct_connect. An application can call ct_con_props to define login parameters.
• A connection can be either synchronous or asynchronous. The OC-Library property CS_NETIO determines whether a connection will be synchronous or asynchronous.
For more information on asynchronous connections, see the Asynchronous Programming topics page.
• The maximum number of open connections per context is determined by the CS_MAX_CONNECT property (set by ct_config). If not explicitly set, the maximum number of connections defaults to a platform-specific value. For information on platform-specific property values, see the SYBASE Interoperability Tools Supplement.
• When a connection attempt is made between a client and a server, there are two ways in which the process can fail (assuming that the system is correctly configured):
- The machine that the server is supposed to be on is running correctly and the network is running correctly.
In this case, if there is no server listening on the specified port, the machine that the server is supposed to be on will signal the client, via a network error, that the connection can't be formed. Regardless of the login timeout value, the connection will fail.
- The machine that the server is on is down.
In this case, the machine that the server is supposed to be on will not respond. Because "no response" is not considered to be an error, the network will not signal the client that an error has occurred. However, if a login timeout period has been set, a timeout error will occur when the client fails to receive a response within the set period.
• To close a connection, an application calls ct_close.
Multiple QUERY Entries in an Interfaces File
• It is possible to set up an interfaces file so that if ct_connect fails to establish a connection with a server, it attempts to establish a connection with an alternate server.
An application can use the ct_connect call:
ct_connect(connection, "MARS", CS_NULLTERM)
to connect to the server MARS. An interfaces file containing an entry for MARS might look like this:

```
#
MARS
  query tcp hp-ether violet 1025
  master tcp hp-ether violet 1025
  console tcp hp-ether violet 1026
#
VENUS
  query tcp hp-ether plum 1050
  master tcp hp-ether plum 1050
  console tcp hp-ether plum 1051
#
```

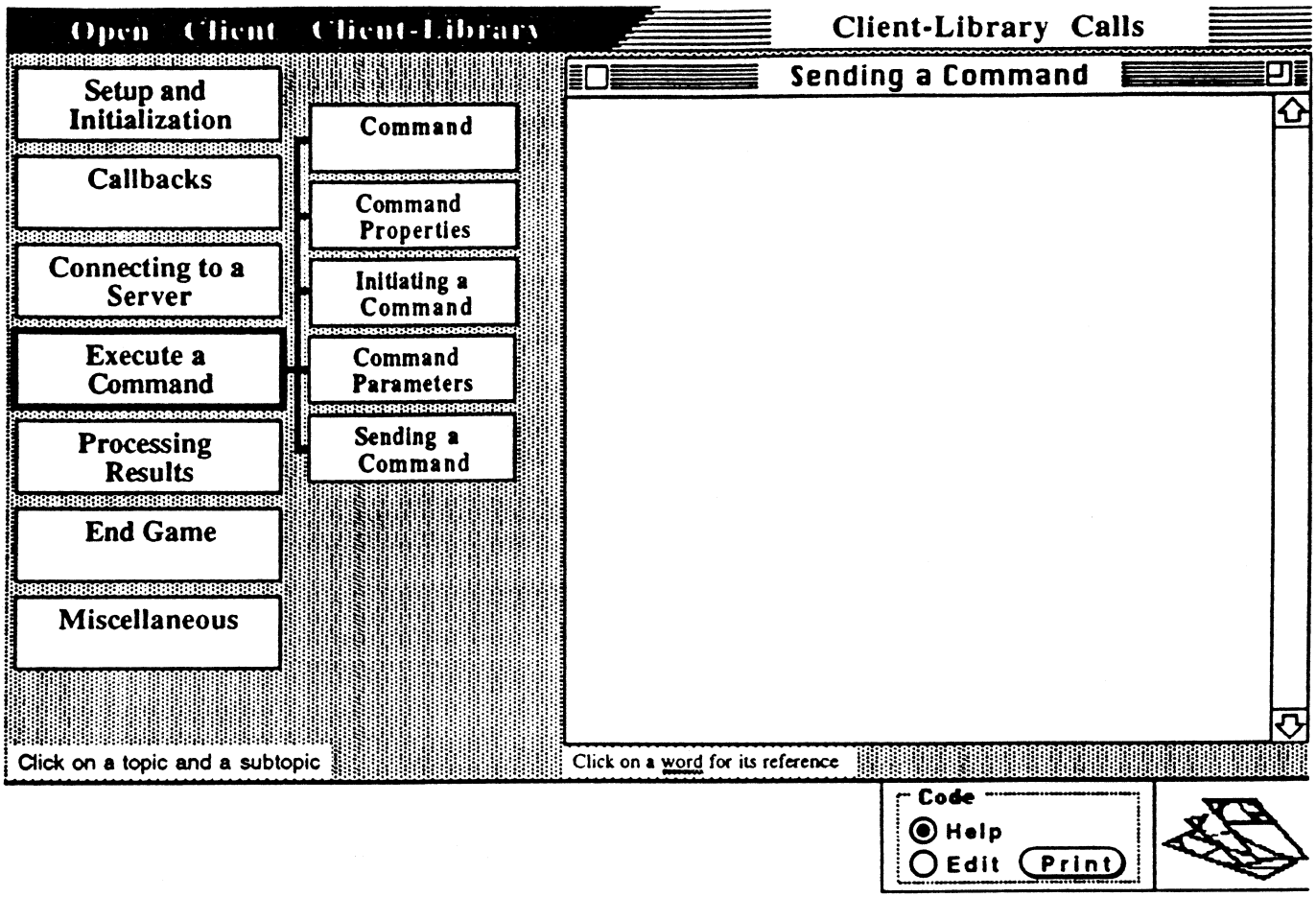
```
NEPTUNE
  query tcp hp-ether mauve 1060
  master tcp hp-ether mauve 1060
  console tcp hp-ether mauve 1061
```
The application is directed to port number 1025 on the machine violet. If MARS is not available, the ct_connect call fails. If the interfaces file has multiple query entries in it for MARS, however, then when the first connection attempt fails, ct_connect will automatically attempt to connect to the next server listed. Such an interfaces file might look like this:
```
#
MARS
  query tcp hp-ether violet 1025
  query tcp hp-ether plum 1050
  query tcp hp-ether mauve 1060
  master tcp hp-ether violet 1025
  console tcp hp-ether violet 1026
#
VENUS
  query tcp hp-ether plum 1050
  master tcp hp-ether plum 1050
  console tcp hp-ether plum 1051
#
NEPTUNE
  query tcp hp-ether mauve 1060
  master tcp hp-ether mauve 1060
  console tcp hp-ether mauve 1061
```
Note that the second query entry under MARS is identical to the query entry under VENUS, and that the third query entry is identical to the query entry under NEPTUNE. If this interfaces file is used, then if the application fails to connect with MARS it will automatically attempt to connect with VENUS. If it fails to connect with VENUS, it will automatically attempt to connect with NEPTUNE. There is no limit on the number of alternate servers that may be listed under a server's interfaces file entry, but each alternate server must be listed in the same interfaces file. Two numbers may be added after the server's name in the interfaces file:
```
#
MARS retries seconds
  query tcp hp-ether violet 1025
  query tcp hp-ether plum 1050
  query tcp hp-ether mauve 1060
  master tcp hp-ether violet 1025
  console tcp hp-ether violet 1026
```
retries represents the number of additional times to loop through the list of query entries if no connection is achieved during the first pass. seconds represents the amount of time, in seconds, that ct_connect will wait at the top of the loop before going through the list again. These numbers are optional. If they are not included, ct_connect will try to connect to each query entry only once. Looping through the list and pausing between loops is useful in case any of the candidate servers is in the process of booting. Multiple query lines can be particularly useful when alternate servers contain mirrored copies of the primary server's databases.

Returns

CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.
CS_PENDING - Asynchronous network I/O is in effect.
CS_BUSY - An asynchronous operation is already pending for this connection.

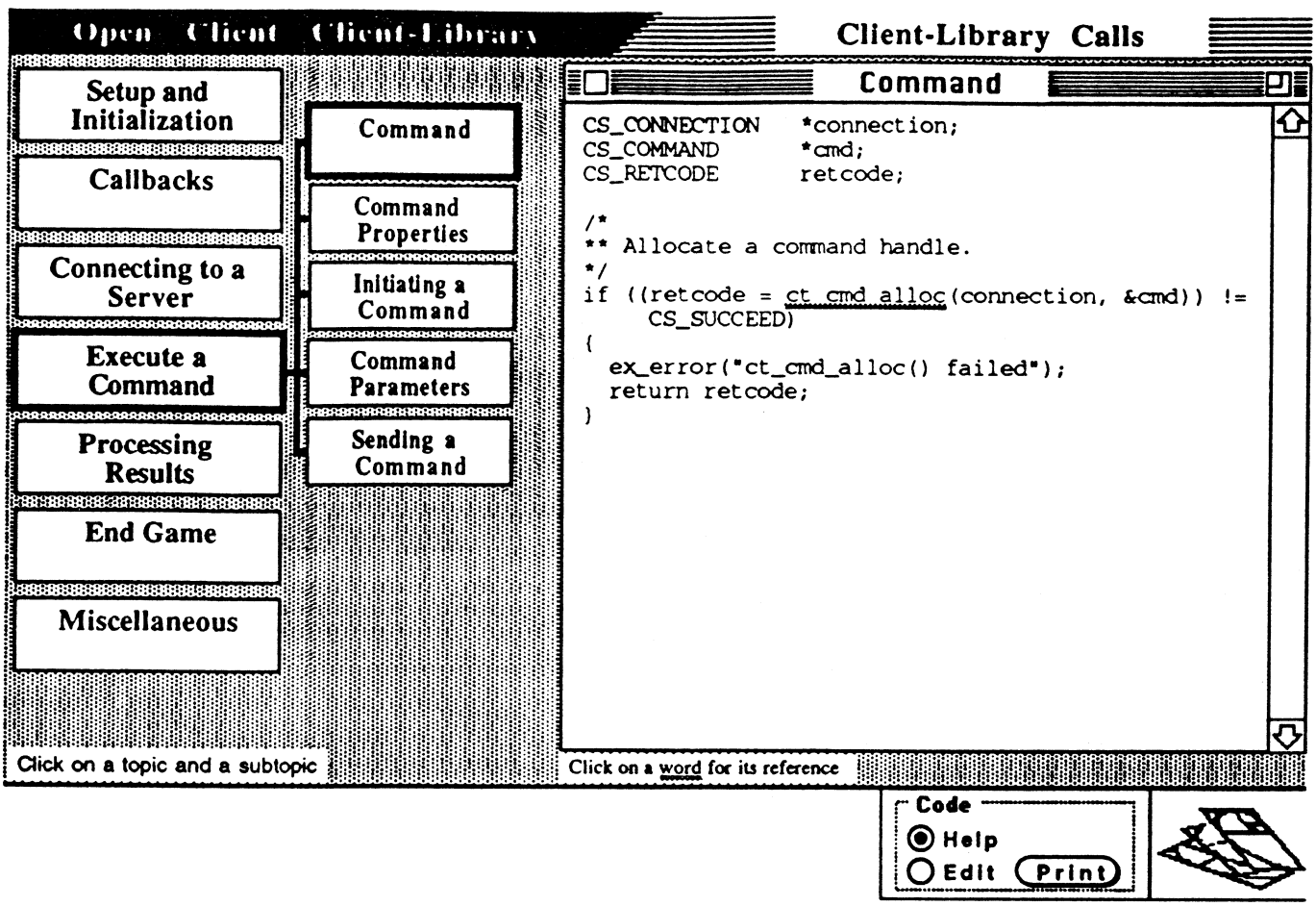
See Also

ct_close, ct_con_alloc, ct_con_drop, ct_con_props, ct_remote_pwd



• An application sends commands to a server using a command structure. Before an application can send a command, it must allocate a command structure and set properties for the command structure, if necessary.

• Once a command structure is available, sending a command to a server includes the following:
- Set the command properties, if needed.
- Initiate the command.
- Define parameters for the command, if needed.
- Send the command.



- An application calls **ct_cmd_alloc** to allocate a command structure.

ct_cmd_alloc

Function
Allocate a CS_COMMAND structure.

Syntax
CS_RETCODE ct_cmd_alloc(connection, cmd_pointer)

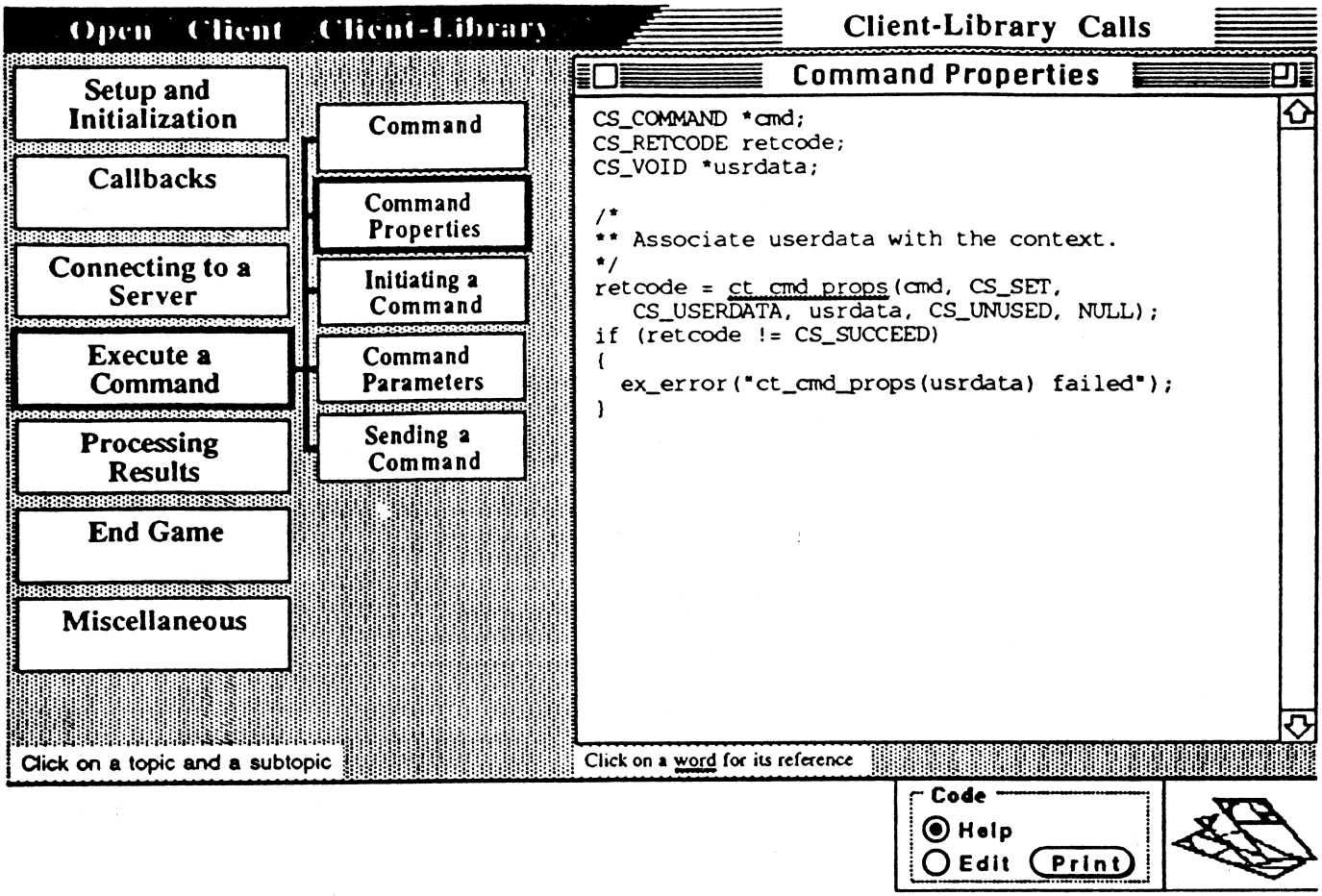
CS_CONNECTION *connection;
CS_COMMAND **cmd_pointer;

Parameters
connection - A pointer to a CS_CONNECTION structure. A CS_CONNECTION structure contains information about a particular client/ server connection.
cmd_pointer - The address of a pointer variable. ct_cmd_alloc sets *cmd_pointer to the address of a newly-allocated CS_COMMAND structure.
In case of error, ct_cmd_alloc sets *cmd_pointer to NULL.

Comments
• ct_cmd_alloc allocates a CS_COMMAND structure.
• A CS_COMMAND structure, also called a "command structure," is a control structure that a Client-Library application uses to send commands to a server and process the results of those commands.
• An application must call ct_con_alloc to allocate a connection structure before calling ct_cmd_alloc to allocate command structures for the connection. However, it is not necessary that the connection structure represent an open connection.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
ct_command, ct_cmd_drop, ct_cmd_props, ct_con_alloc



• An application calls **ct_cmd_props** to set, retrieve, or clear command structure properties.

•Command structure properties determine aspects of Client-Library behavior at the command structure level. For example, the CS_HIDDEN_KEYS property determines whether or not Client-Library exposes any hidden keys that are returned as part of a result set.

• When a command structure is allocated, it picks up default property values from its parent connection.

• Some examples of command properties are:
- CS_CUR_NAME : The name of the cursor as defined in a cursor declare call.
- CS_PARENT_HANDLE : The address of the parent connection structure.
- CS_USERDATA : User-allocated data.

ct_cmd_props

Function
Set or retrieve command structure properties.

Syntax
CS_RETCODE ct_cmd_props(cmd, action, property, buffer, buflen, outlen)

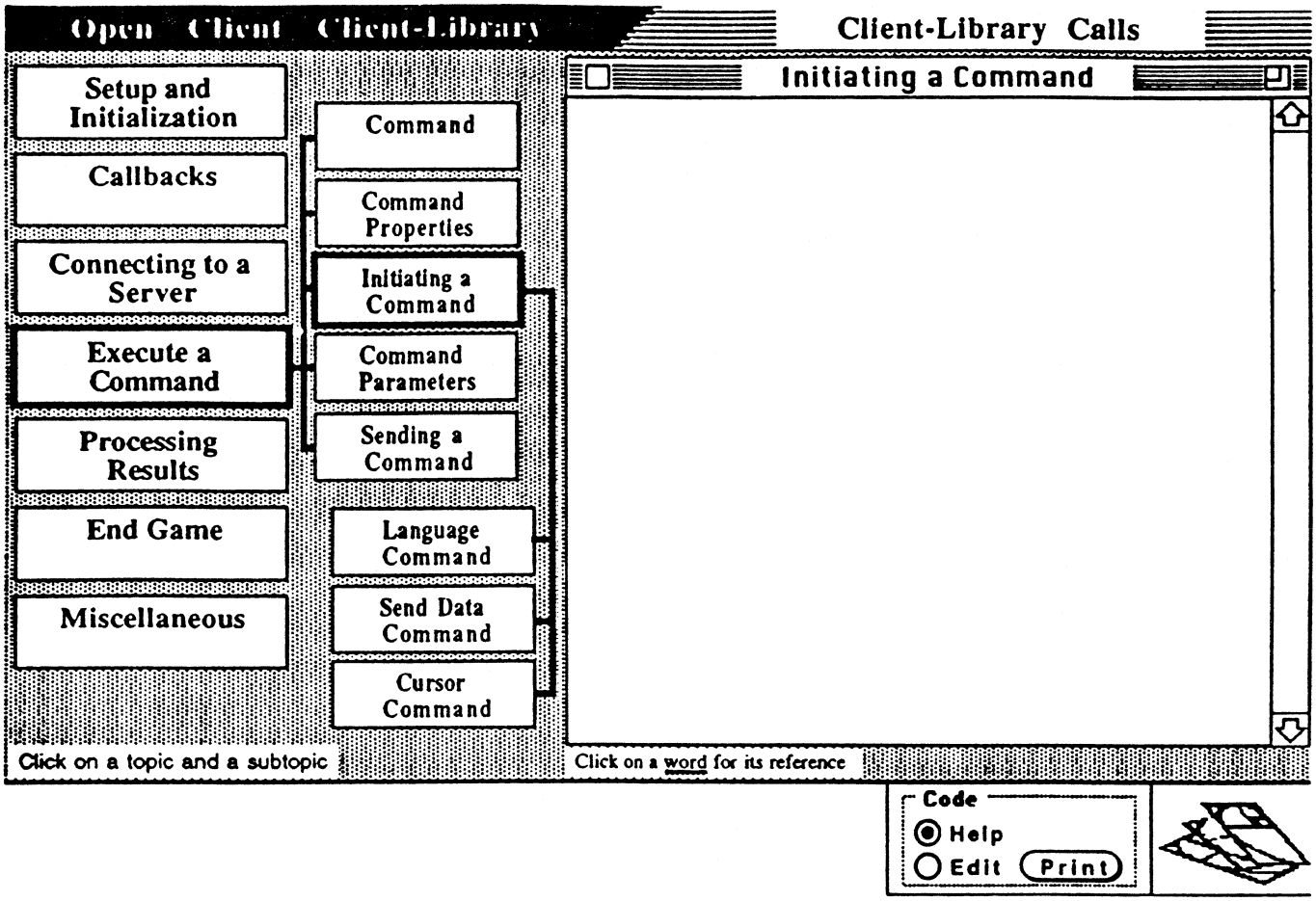
```
CS_COMMAND *cmd;
CS_INT action;
CS_INT property;
CS_VOID *buffer;
CS_INT buflen;
CS_INT *outlen;
```

Parameters
cmd - A pointer to the CS_COMMAND structure managing a client/ server operation.
action - An integer variable which has been set to one of the following symbolic values:
CS_SET : Sets the value of the property.
CS_GET : Retrieves the value of the property.
CS_CLEAR : Clears the value of the property by resetting it to its Client-Library default value.
property - The symbolic name of the property whose value is being set or retrieved. The Properties topics page lists Client-Library properties.
buffer - If a property value is being set, buffer points to the value to use in setting the property. If a property value is being retrieved, buffer points to the space in which ct_cmd_props will place the requested information.
buflen - The length, in bytes, of *buffer. If a property value is being set and the value in *buffer is null-terminated, pass buflen as CS_NULLTERM.
outlen - A pointer to an integer variable. outlen is not used if a property value is being set and should be passed as NULL. If a property value is being retrieved and outlen is supplied, ct_cmd_props sets *outlen to the length, in bytes, of the requested information. If the information is larger than buflen bytes, an application can use the value of *outlen to determine how many bytes are needed to hold the information.

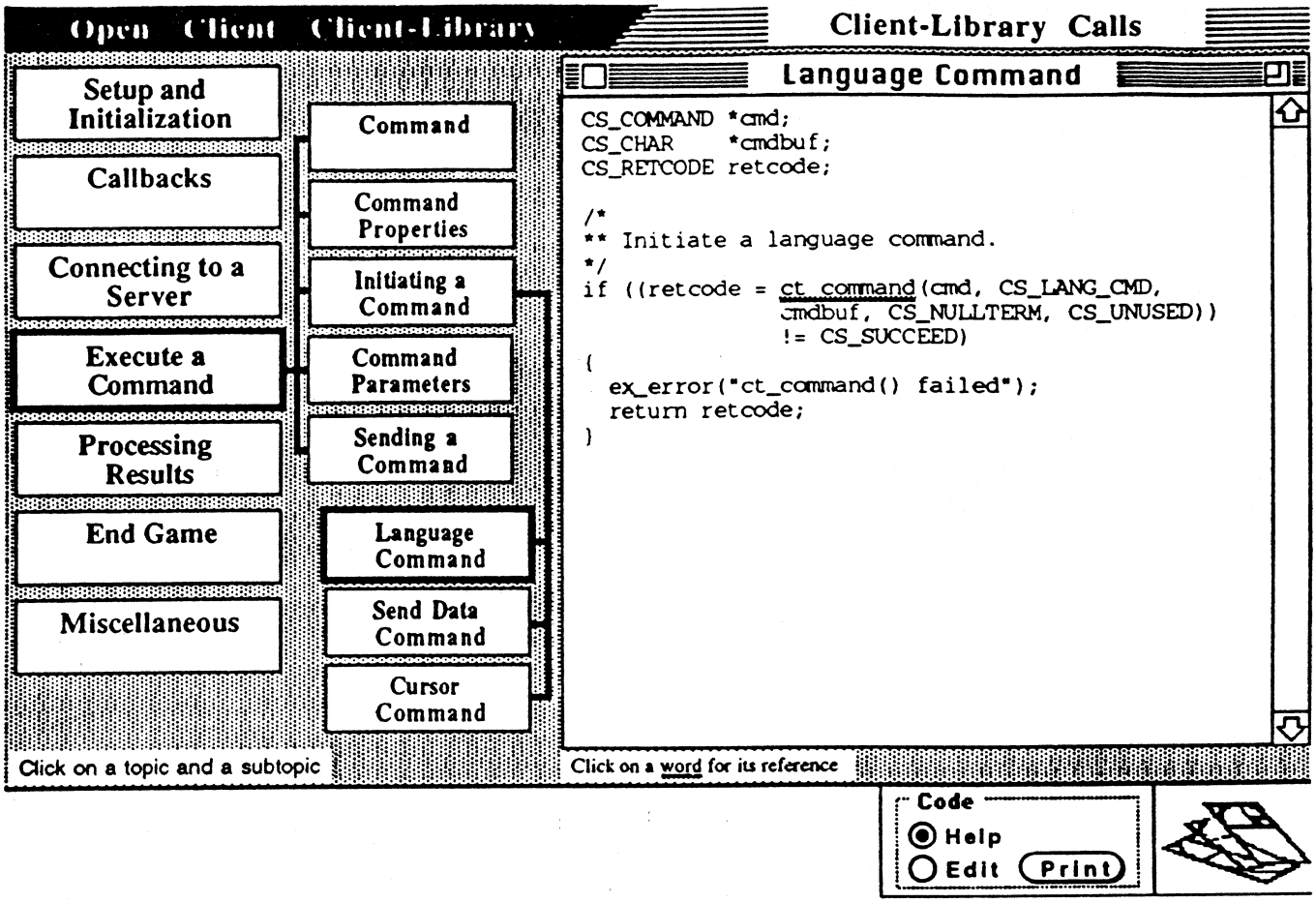
Comments
• ct_cmd_props sets or retrieves the values of command structure properties.
• Command structure properties affect the behavior of an application at the command structure level.
• See the Properties topics page for more information on properties.
• An application can use ct_cmd_props to set or retrieve the following properties:
CS_CUR_ID : The cursor's identification number.
CS_CUR_NAME : The cursor's name, as defined in an application's ct_cursor(CS_CUSOR_DECLARE) call.
CS_CUR_ROWCOUNT : The current value of cursor rows. Cursor rows is the number of rows returned to Client-Library per internal fetch request.
CS_CUR_STATUS : The cursor's status.
CS_HIDDEN_KEYS : Whether or not to expose hidden keys.
CS_PARENT_HANDLE : The address of the command structure's parent connection.
CS_USERDATA : User-allocated data.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
ct_config, ct_cmd_alloc, ct_con_props, ct_res_info



• An application can send several types of commands to a server:
- Call **ct_command** to initiate a language, message, package, RPC, or send-data command.
- Call **ct_cursor** to initiate a cursor command.
- Call **ct_dynamic** to initiate a dynamic SQL command.



• Language commands contain character strings that represent commands in a server's own language. Transact-SQL is an example of Sybase SQL Server's command language.

• A language command can contain more than one server command.

• If the language command contains variables, **ct_param** is used to pass values for these variables.

ct_command

Function
Initiate a language, package, RPC, message, or send-data command.

Syntax
CS_RETCODE ct_command(cmd, type, buffer, buflen, option)

```
CS_COMMAND *cmd;
CS_INT type;
CS_VOID *buffer;
CS_INT buflen;
CS_INT option;
```

Parameters
cmd - A pointer to the CS_COMMAND structure managing a client/ server operation.
type - The type of command to initiate. The following lists the symbolic values that are legal for type:
CS_LANG_CMD : A language command.
CS_MSG_CMD : A message command.
CS_PACKAGE_CMD : A package command.
CS_RPC_CMD : A remote procedure call command.
CS_SEND_DATA_CMD : A send-data command.
CS_SEND_BULK_CMD : A SYBASE internal send-bulk-data command.
buffer - A pointer to data space.
buflen - The length, in bytes, of *buffer.
option - The option associated with this command, if any. Currently, RPC (remote procedure call), send-data, and send-bulk-data commands take options. For all other types of commands, pass options as CS_UNUSED. The following lists the symbolic values that are legal for option:

Comments
• ct_command initiates a language, message, package, RPC (remote procedure call), send-data, or send-bulk-data command. Initiating a command is the first step in sending it to a server.
• Sending a command to a server is a three step process. To send a command to a server, an application must:
- Initiate the command by calling ct_command, ct_cursor, or ct_dynamic. These routines set up internal structures that are used in developing a command stream to send to the server.
- Pass parameters for the command. Most applications pass parameters by calling ct_param once for each parameter that the command requires, but it is also possible to pass parameters for Dynamic SQL commands by using ct_dyndesc.
Not all commands require parameters. For example, a remote procedure call command may or may not require parameters, depending on the stored procedure being called.
- Call ct_send to send the command stream to the server.
• An application can call ct_cancel with type as CS_CANCEL_ALL to clear a command that has been initiated but not yet sent.

Language Commands
• Language commands contain character strings that represent commands in a server's own language. For example, the following language command contains a Transact-SQL command:
ct_command(cmd, CS_LANG_CMD, "select * from authors ", CS_NULLTERM,CS_UNUSED);
• The character string that makes up a language command can contain more than one server command. For example, the following language command contains three Transact-SQL commands:
ct_command(cmd, CS_LANG_CMD, "use pubs2 select * from titles select * from authors ", CS_NULLTERM,CS_UNUSED);
• A language command can be in any language, so long as the server to which it is directed can understand it. SQL Server understands Transact-SQL, but an Open Server application constructed with SYBASE Server-Library can be written to understand any language.
• If the language command string contains variables, an application can pass values for these variable by calling ct_param once for each variable that the language string contains.
• Transact-SQL command variables must begin with a colon, the character ":"

Message Commands
• Message commands and results provide a way for clients and servers to communicate specialized information to one another.

• If a message requires parameters, the application can call ct_param once for each parameter that the message requires.

Package Commands
• A package command instructs an IBM DB/2 database server to execute a package. A package is similar to a remote procedure. It contains precompiled SQL statements that are executed as a unit when the package is invoked.
• If the package requires parameters, the application can call ct_param once for each parameter that the package requires.

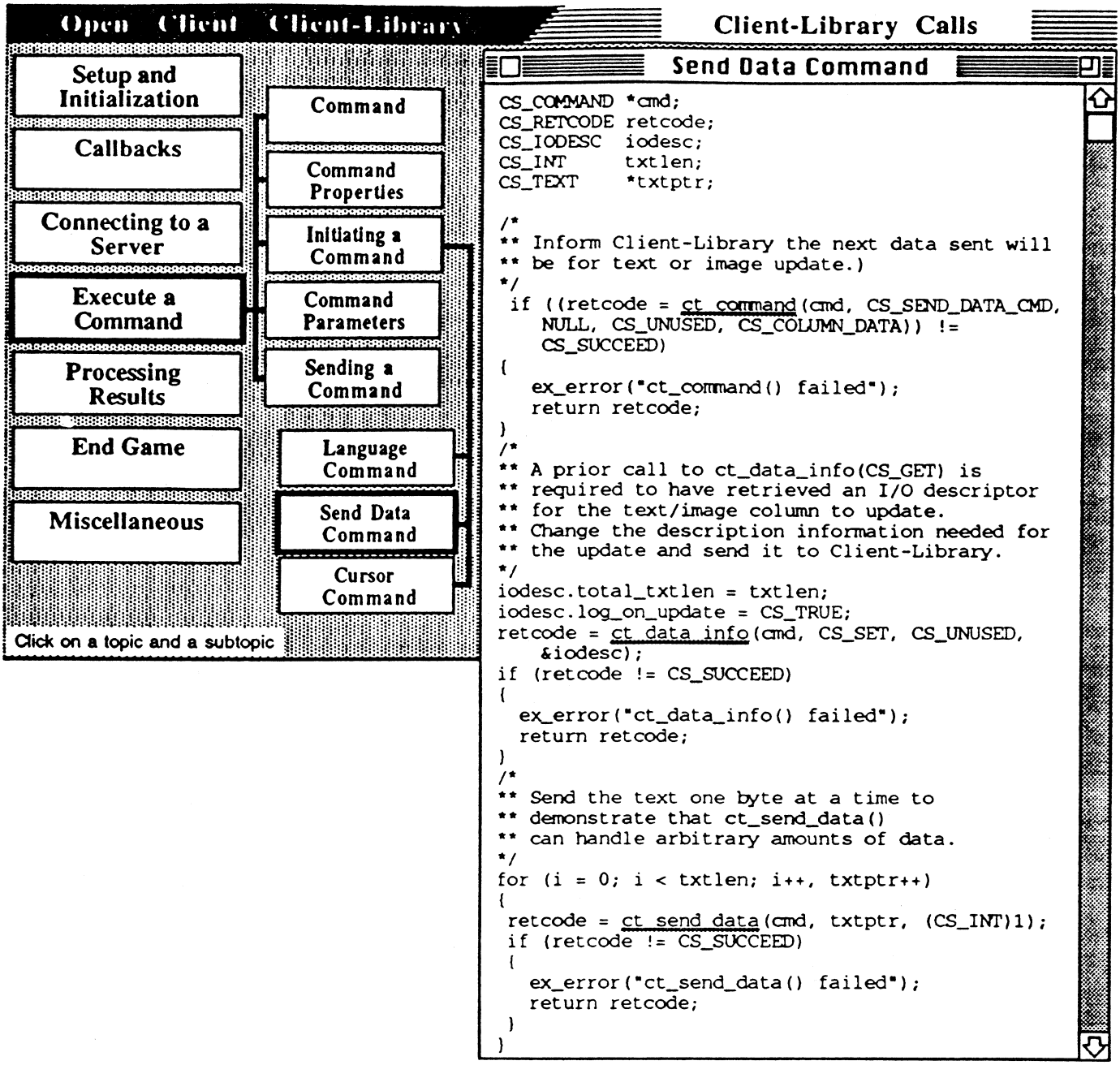
RPC (remote procedure call) Commands
• An RPC (remote procedure call) command instructs a server to execute a stored procedure either on this server or a remote server.
• An application can call a stored procedure in two ways: by executing a Transact-SQL language command (an execute statement), or by executing an RPC command. See the Remote Procedures topics page for a discussion of the differences between these techniques.
• If an application is using an RPC command to execute a stored procedure that requires parameters, the application can call ct_param once for each parameter the stored procedure requires.
• After sending an RPC command with ct_send, an application can process the stored procedure's results with ct_results and ct_fetch. ct_results and ct_fetch are used to process both the result rows generated by the stored procedure and the return parameters and status from the procedure, if any.

Send-Data Commands
• An application uses a send-data command to write large amounts of text or image data to a server.
• An application typically calls:
- ct_command to initiate the send-data command.
- ct_data_info to set the I/O descriptor for the operation.
- ct_send_data to write the value, in chunks, to the data stream.
- ct_send to send the command to the server.
• For more information on writing text or image values, see the Text and Image topics page.
• Internally, Sybase uses send-data commands as part of its implementation of Client-Library's bulk copy routines.

Send-Bulk-Data Commands
• Internally, Sybase uses send-bulk-data commands as part of its implementation of Client-Library's bulk copy routines.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
ct_cmd_alloc, ct_cursor, ct_dynamic, ct_param, ct_send



- An application uses a send-data command to write large amounts of text or image data to a server.

- An application typically calls:
- **ct_command** to initiate the send-data command.
- **ct_data_info** to set the I/O descriptor for the operation.
- **ct_send_data** to write the value, in chunks, to the data stream.
- **ct_send to send** the command to the server.

ct_data_info
Function
Define or retrieve a data I/O descriptor structure.
Syntax
CS_RETCODE ct_data_info(cmd, action, colnum, iodesc)

CS_COMMAND *cmd;
CS_INT action;
CS_INT colnum;
CS_IODESC *iodesc;
Parameters
cmd - A pointer to the CS_COMMAND structure managing a client/ server operation.
action - An integer variable which has been set to one of the following symbolic values:
Value of *action:
ct_data_info:
CS_SET
Defines an I/O descriptor.
CS_GET
Retrieves an I/O descriptor.
Table 3-44: Values for action (ct_data_info)
colnum - The number of the text or image column whose I/O descriptor is being retrieved.
colnum is used only when action is CS_GET.
colnum refers to the select-list id of the text or image column. The first column in a select statement's select-list is column number 1, the second number 2, and so forth. An application must select a text or image column before it can update the column.
colnum must represent a text or image column.
iodesc - A pointer to a CS_IODESC structure. A CS_IODESC structure contains information describing text or image data.
Comments
• ct_data_info defines or retrieves a CS_IODESC, also called an "I/O descriptor structure," for a text or image column.
• An application defines an I/O descriptor before updating a text or image value, setting the total_txtlen field of the CS_IODESC to the total length, in bytes, of the new text or image value.
• An application retrieves an I/O descriptor after calling ct_get_data to retrieve a text or image column that it plans to update at a later time.
• When defining an I/O descriptor for a text or image update, an application must set the log_on_update field in the CS_IODESC to indicate whether or not the server should log the update.
• A successful text or image update generates a parameter result set that contains the new text timestamp for the text or image value. If an application plans to update the text or image value a second time, it must save this new text timestamp and copy it into the CS_IODESC for the value before calling ct_data_info to define the CS_IODESC for the update operation.
• It is illegal to call ct_data_info to retrieve the I/O descriptor for a column before calling ct_get_data to retrieve the column's value.
Returns
Returns:
To Indicate:
CS_SUCCEED
The routine completed successfully.
CS_FAIL
The routine failed.
CS_BUSY
An asynchronous operation is already pending for this connection. For more information, see the Asynchronous Programming topics page.
Table 3-45: Return Values (ct_data_info)
The most common reason for a ct_data_info failure is...
See Also
ct_get_data, ct_send_data, Text and Image

ct_send_data

Function
Send a chunk of text or image data to the server.

Syntax
CS_RETCODE ct_send_data(cmd, buffer, buflen)

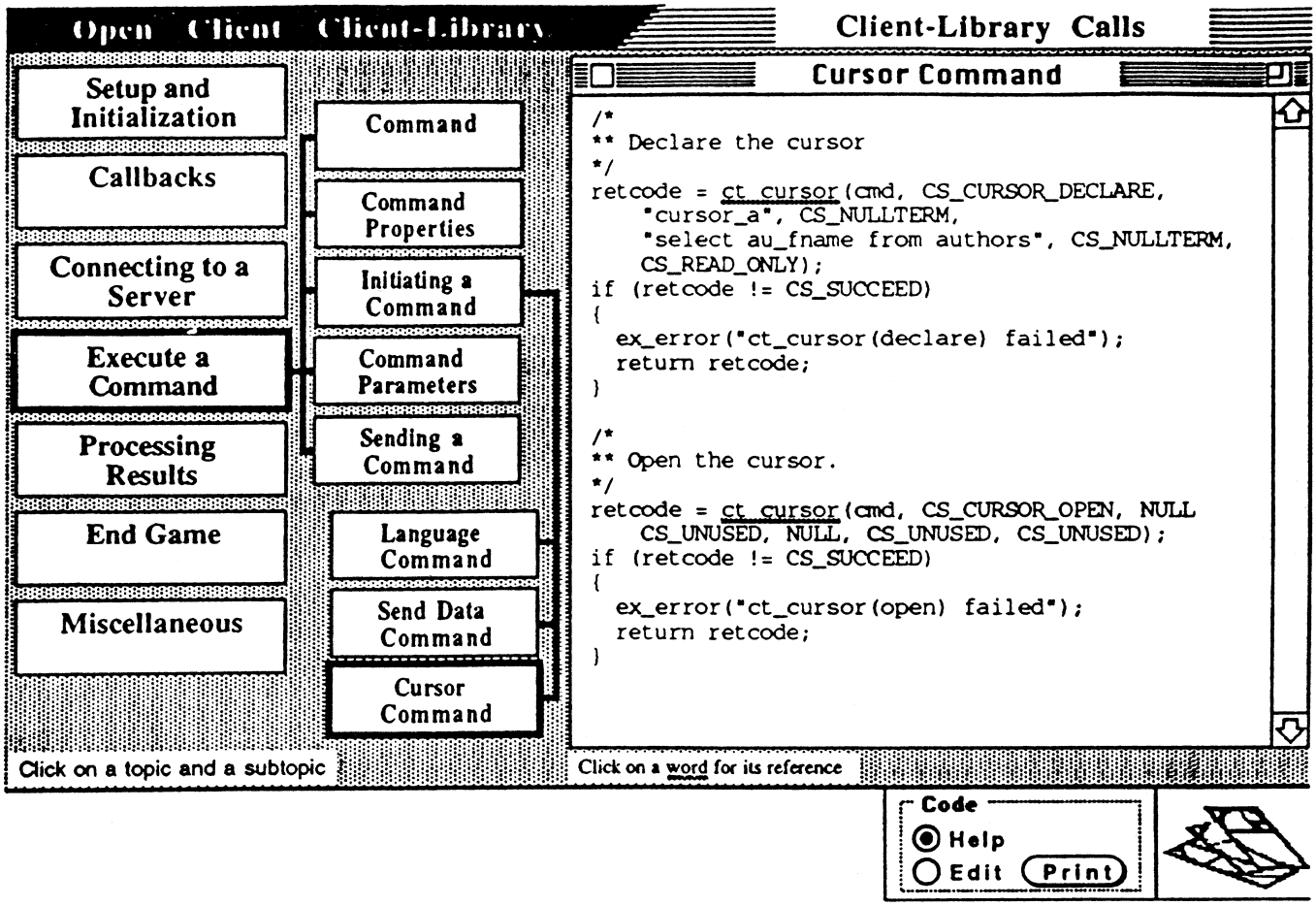
CS_COMMAND *cmd;
CS_VOID *buffer;
CS_INT buflen;

Parameters
cmd - A pointer to the CS_COMMAND structure managing a client/ server operation.
buffer - A pointer to the value to write to the server.
buflen - The length, in bytes, of *buffer.
If the value in *buffer is null-terminated, pass buflen as CS_NULLTERM.

Comments
• ct_send_data sends a chunk of text or image data to the server.
• An application can use ct_send_data to write a text or image value to a database column. This writing operation is actually an update; that is, the column must have a value when ct_send_data is called to write a new value. This is because ct_send_data uses text timestamp information when writing to the column, and a column does not have a valid text timestamp until it contains a value. The value contained in the text or image column can be NULL, but the NULL must be entered explicitly with the SQL update statement.
• Before calling ct_send_data, an application must call ct_command to initiate the send-data command and ct_data_info to set the current I/O descriptor. This I/O descriptor, or CS_IODESC structure, describes the column and its new value:
- The textptr field of the CS_IODESC identifies the target column.
- The total_txtlen field of the CS_IODESC indicates the total length, in bytes, of the column's new value. An application must call ct_send_data in a loop to write exactly this number of bytes before calling ct_send to indicate the end of the text or image update operation.
• After calling ct_send, the application must call ct_results to determine the result of the update.
• A text or image update operation is equivalent to a language command containing a Transact-SQL update statement.
• The command space identified by cmd must be idle before a text or image update operation is initiated. A command space is idle if there are no active commands, pending results, or open cursors in the space.
• For more information on writing a text or image value, see the Text and Image topics page.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.
CS_CANCELED - The send data operation was canceled.
CS_PENDING - Asynchronous network I/O is in effect.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
ct_data_info, ct_get_data, Text and Image



• ct_cursor initiates a cursor command. Initiating a command is the first step in sending it to a server.

• An application can "batch" together commands. The application calls **ct_cursor** once for each command being initialized. Batched commands must be initialized in the logical order: declare, set cursor rows, open. Batching cursor commands reduces network traffic and can improve application performance.

• Cursor commands include:
- **Declare** : Declaring a cursor is equivalent to associating the cursor name with SQL text. The cursor can be declared for readonly or update.
- **Open** : A cursor open command executes the body of the cursor, generating a cursor result set. The result set can be accessed using ct_results, ct_bind, and ct_fetch.
- **Rows** : A cursor rows command sets the number of rows returned for each fetch request.
- **Update** : A cursor update command defines new column values for the current cursor row. These new values are used to update an underlying server table.
- **Close** : A close cursor command discards the cursor result set
- **De-allocate** : A cursor de-allocate command de-allocates a closed cursor.
- **Delete** : A cursor delete command deletes the current cursor row from the cursor result set. The delete is propagated back to the underlying server tables.

ct_cursor

Function
Initiate a cursor command.

Syntax
CS_RETCODE ct_cursor(cmd, type, name, namelen, text, textlen, option)

```
CS_COMMAND *cmd;
CS_INT type;
CS_CHAR *name;
CS_INT namelen;
CS_CHAR *text;
CS_INT textlen;
CS_INT option;
```

Parameters
cmd - A pointer to the CS_COMMAND structure managing a client/ server operation.
type - The type of command to initiate. The chart in the Summary of Parameters section lists the symbolic values that are legal for type.
name - A pointer to the name associated with the cursor command, if any. The chart in the Summary of Parameters section indicates which types of commands require names.
namelen - The length, in bytes, of *name. If *name is null-terminated, pass namelen as CS_NULLTERM. If name is NULL pass namelen as CS_UNUSED.
text - A pointer to the text associated with the cursor command, if any. The chart in the Summary of Parameters section indicates which commands require text and what that text must be.
textlen - The length, in bytes, of *text. If *text is null-terminated, pass textlen as CS_NULLTERM. If text is NULL, pass textlen as CS_UNUSED.
option - The option associated with this command, if any. The chart in the Summary of Parameters section indicates which commands take an option and what that option can be.

Comments
• ct_cursor initiates a cursor command. Initiating a command is the first step in sending it to a server. Cursor commands include commands to declare, open, set rows for, close, update-positioned, delete-positioned, and de-allocate a cursor.
• Sending a command to a server is a three step process. To send a command to a server, an application must:
- Initiate the command by calling ct_command, ct_cursor, or ct_dynamic. These routines set up internal structures that are used in developing a command stream to send to the server.
- Pass parameters for the command. Most applications pass parameters by calling ct_param once for each parameter that the command requires, but it is also possible to pass parameters for Dynamic SQL commands by using ct_dyndesc.
Not all commands require parameters. For example, a remote procedure call command may or may not require parameters, depending on the stored procedure being called.
- Call ct_send to send the command stream to the server.

Batching Cursor Commands
• An application can "batch" together commands to declare, set rows for, and open a cursor. To do this, the application calls ct_cursor once for each command being initialized, calls ct_param if required, and calls ct_send once to send the command batch to the server. For example:

```
ct_cursor(cmd, CS_CURSOR_DECLARE, "au_cursor", CS_NULLTERM,
  "select au_id, au_lname from authors", CS_NULLTERM, CS_FOR_UPDATE);
ct_cursor(cmd, CS_CURSOR_ROWS, NULL,CS_UNUSED,  NULL, CS_UNUSED, 50);
ct_cursor(cmd, CS_CURSOR_OPEN, NULL, CS_UNUSED,  NULL, CS_UNUSED, CS_UNUSED);
ct_send(cmd);
```

Batched commands must be initialized in the logical order: declare, set cursor rows, open. Batching cursor commands reduces network traffic and can improve application performance.

Cursor Close
• A close cursor command throws away the cursor result set that was generated when the cursor was opened.
• An application can re-open a closed cursor.
• To initiate a command to both close and de-allocate a cursor, call ct_cursor with type as CS_CURSOR_CLOSE and option as CS_DEALLOC.

Cursor De-allocate

• A cursor de-allocate command de-allocates a cursor. If a cursor has been de-allocated, it cannot be re-opened.
• An application cannot de-allocate an open cursor.

Cursor Declare
• Declaring a cursor is equivalent to associating the cursor name with SQL text. This SQL text is called the body of the cursor. When a cursor is opened, the body of the cursor is executed, generating a cursor result set.
• The SQL text associated with a cursor can contain host variables. If this is the case, an application must call ct_param at cursor declare time to define the variables' formats.
• If option is CS_FOR_UPDATE, the cursor is declared for update. This means that the cursor can be used to change values in the underlying server tables.
- To indicate that all of the columns returned by a cursor are for update, the ct_cursor call declaring the cursor for update must be followed immediately by an ct_send call sending the command stream to the server.
For example, to indicate that all columns returned by a cursor are for update:
```
ct_cursor(cmd, CS_CURSOR_DECLARE, "au_cursor", CS_NULLTERM, "select * from authors",
   CS_NULLTERM, CS_FOR_UPDATE);
ct_send(cmd);
```
- To indicate that only some of the columns returned by a cursor are for update, the ct_cursor call declaring the cursor for update must be immediately followed by one or more ct_param calls indicating the update columns.
For example, to indicate that au_id and au_lname are for update:
```
ct_cursor(cmd, CS_CURSOR_DECLARE,  "au_cursor", CS_NULLTERM, "select * from authors",
  CS_NULLTERM, CS_FOR_UPDATE);
format.status = CS_UPDATECOL;
ct_param(cmd, &format, "au_id", CS_NULLTERM, CS_UNUSED);
format.status = CS_UPDATECOL;
ct_param(cmd, &format, "au_lname", CS_NULLTERM, CS_UNUSED);
ct_send(cmd);
```
• If option is CS_READ_ONLY, the cursor is declared read-only. This means that the cursor cannot be used to change values in the underlying server tables.
• The SQL text associated with a cursor can be a command to execute a stored procedure.
For example:
```
ct_cursor (cmd, CS_CURSOR_DECLARE, "mycursor",
  CS_NULLTERM, "execute my_proc",
  CS_NULLTERM, CS_UNUSED);
ct_send(cmd);
```
In this case, the body of the cursor is really the SQL text that makes up the stored procedure. If the stored procedure takes parameters, an application can pass values for them by calling ct_param at cursor open time.
• An alternate way to declare a cursor is through a language command. These types of cursors are called "language-based cursors". To declare a language-based cursor, an application calls ct_command and ct_send to send a Transact-SQL language command declaring a cursor to a server. This method, of course, can only be used with servers that understand Transact-SQL.
Note that an application cannot use ct_cursor commands on a language-based cursor.

Cursor Delete
• A cursor delete command deletes the current cursor row from the cursor result set. The delete is propagated back to the underlying server tables.

Cursor Open
• A cursor open command executes the body of the cursor, generating a cursor result set. After a cursor is opened, an application can access the cursor rows using ct_results, ct_bind, and ct_fetch.
• Some cursors require input parameter values at cursor open time. An application can pass input parameter values for a cursor open command by calling ct_param after calling ct_cursor. A cursor open command requires parameters:
- If the body of the cursor is a SQL text string that contains host variables.
- If the body of the cursor is a stored procedure that requires input parameter values.

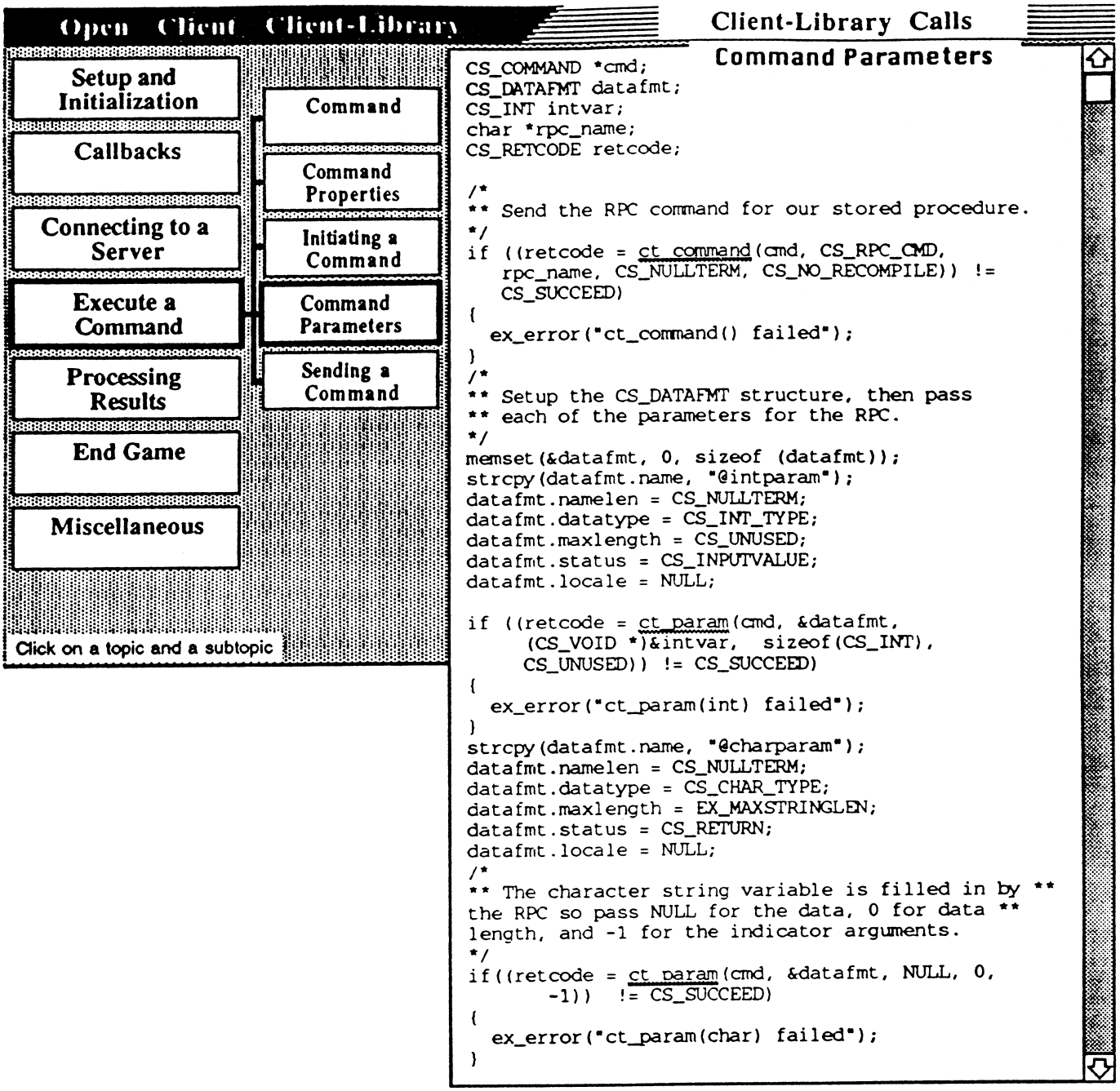
Cursor Rows
• A cursor rows command sets the number of rows that the server returns to Client-Library per each internal fetch request. Note that this is not the number of rows returned to an application per ct_fetch call. The number of rows returned to an application per ct_fetch call is determined by the value of the count field in the CS_DATAFMT structures used in binding the cursor result columns.

• An application can only set cursor rows before opening a cursor.

Cursor Update
• A cursor update command defines new column values for the current cursor row. These new values are used to update an underlying server table.
• When updating a SQL Server table, an application must specify the name of the table to update twice: once as the value of the *name parameter and a second time in the update statement (update tableneme...).
• A cursor update command updates only a single table.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
Cursors, ct_cmd_alloc, ct_keydata, ct_param, ct_results, ct_send



- The following types of commands can take parameters:
- A language command, when the command text contains variables.
- An RPC command, when the stored procedure takes parameters.
- A cursor declare command, when the body of the cursor contains host variables or when some (but not all) of the cursor's columns are for update.
- A cursor open command.
- A message command.
- A dynamic SQL execute command.

- An application calls **ct_param** once for each parameter a command requires.

ct_param

Function
Define a command parameter.

Syntax
CS_RETCODE ct_param(cmd, datafmt, data, datalen, indicator);

CS_COMMAND *cmd;
CS_DATAFMT *datafmt;
CS_VOID *data;
CS_INT datalen;
CS_SMALLINT indicator;

Parameters
cmd - A pointer to the CS_COMMAND structure managing a client/ server operation.
datafmt - A pointer to a CS_DATAFMT structure that describes the parameter.
For information on how to set these fields for specific uses of ct_param, see the charts in the Comments section
data - The address of the parameter data.
To indicate a parameter with a null value, pass indicator as -1. If indicator is -1, data and datalen are ignored. For example, an application might pass parameters with null values to a stored procedure that assigns default values to NULL input parameters.
datalen - The length, in bytes, of the parameter data.
indicator - An integer variable used to indicate a parameter with a null value. To indicate a parameter with a null value, pass indicator as -1. If indicator is -1, data and datalen are ignored.

Comments
• An application may need to call ct_param:
- To identify update columns for a cursor declare command.
- To define host variable formats for a cursor declare command.
- To pass input parameter values for a cursor open, cursor update, Dynamic SQL execute, language, message, or RPC command.
For specific information on these uses, see the following sections, ''Identifying Update Columns for a Cursor Declare Command'', ''Defining Host Variable Formats'', and ''Passing Input Parameter Values''.
• An application calls ct_command to initiate a language, RPC or message command, calls ct_cursor to initiate a cursor declare or cursor open command, and calls ct_dynamic to initiate a Dynamic SQL execute command.

Identifying Update Columns for a Cursor Declare Command
• An application needs to identify update columns for a cursor declare command if some, but not all, of the columns are "for update." Update columns can be used to change values in underlying database tables.
• If all of the cursor's columns are for update, an application does not need to call ct_param to specify them individually.
• To identify an update column for a cursor declare command, an application calls ct_param with datafmtÆstatus as CS_UPDATECOL and *data as the name of the column

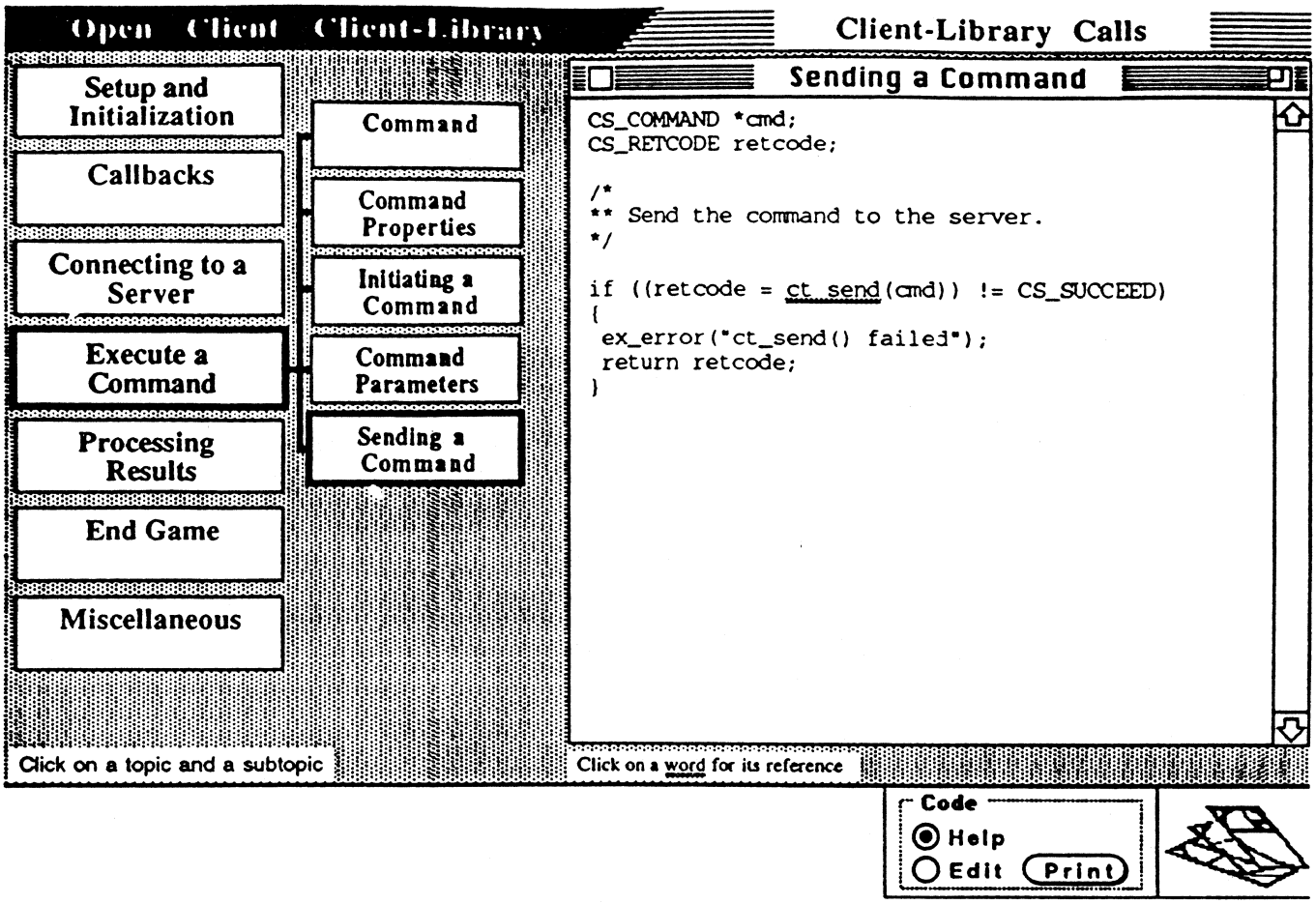
Defining Host Variable Formats
• An application needs to define host variable formats for cursor declare commands, when the body of the cursor being declared is a SQL string that contains host variables.
• To do this, an application calls ct_param with datafmtÆstatus as CS_INPUTVALUE, datafmtÆdatatype as the datatype of the host variable, and datafmtÆlocale as a pointer to the CS_LOCALE containing locale information for the variable, if any.
• When defining host variable formats, the variables can either be named or unnamed. If one variable is named, all variables must be named. If variables are not named, they are interpreted positionally.
• The following table lists the fields in *datafmt that are used when defining host variable formats:
name : The name of the host variable.
namelen : The length, in bytes, of name, or 0 to indicate an unnamed parameter.
datatype : The datatype of the host variable.
status : CS_INPUTVALUE
All other fields : Are ignored.

Passing Input Parameter Values

• An application may need to pass input parameter values:
- For cursor open commands.
- For cursor update commands.
- For Dynamic SQL execute commands.
- For language commands.
- For message commands.
- For RPC commands.
• When passing input parameter values, parameters can either be named or unnamed. If one parameter is named, all parameters must be named. If parameters are not named, they are interpreted positionally.
• Cursor open commands require input parameter values when:
- The body of the cursor is a SQL text string containing host variables.
- The body of the cursor is a stored procedure that requires parameters. In this case, *datafmt->status can be either CS_RETURN, to indicate that the a return parameter, or CS_INPUTVALUE, to indicate a non-return parameter.
• Cursor update commands require input parameter values when the SQL text representing the update command contains host variables.
• Dynamic SQL execute commands require input parameter values when the prepared statement being executed contains host variables.
• Language commands require input parameter values when the text of the language command contains host variables.
• Message commands require input parameters values when the message takes parameters.
• RPC commands require input parameter values when the stored procedure being executed takes parameters.
• The following table lists the fields in *datafmt that are used when passing input parameter values:
name : The name of the parameter.
namelen : The length, in bytes, of name, or 0 to indicate an unnamed parameter.
datatype : The datatype of the input parameter value.
maxlength : When passing return parameters for RPC commands, maxlength represents the maximum length, in bytes, of data to be returned for this parameter. maxlength is not used when passing input parameter values for other types of commands.
status : CS_RETURN when passing return parameters for RPC commands; otherwise CS_INPUTVALUE.
locale : A pointer to a CS_LOCALE structure containing locale information for the host variable, or NULL if locale information is not required.
All other fields : Are ignored.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
ct_command, ct_cursor, ct_dynamic, ct_send



• After a command has been initiated and its parameters (if any) defined, an application calls **ct_send** to send the command to the server.

ct_send

Function
Send a command to the server.

Syntax
CS_RETCODE ct_send(cmd)

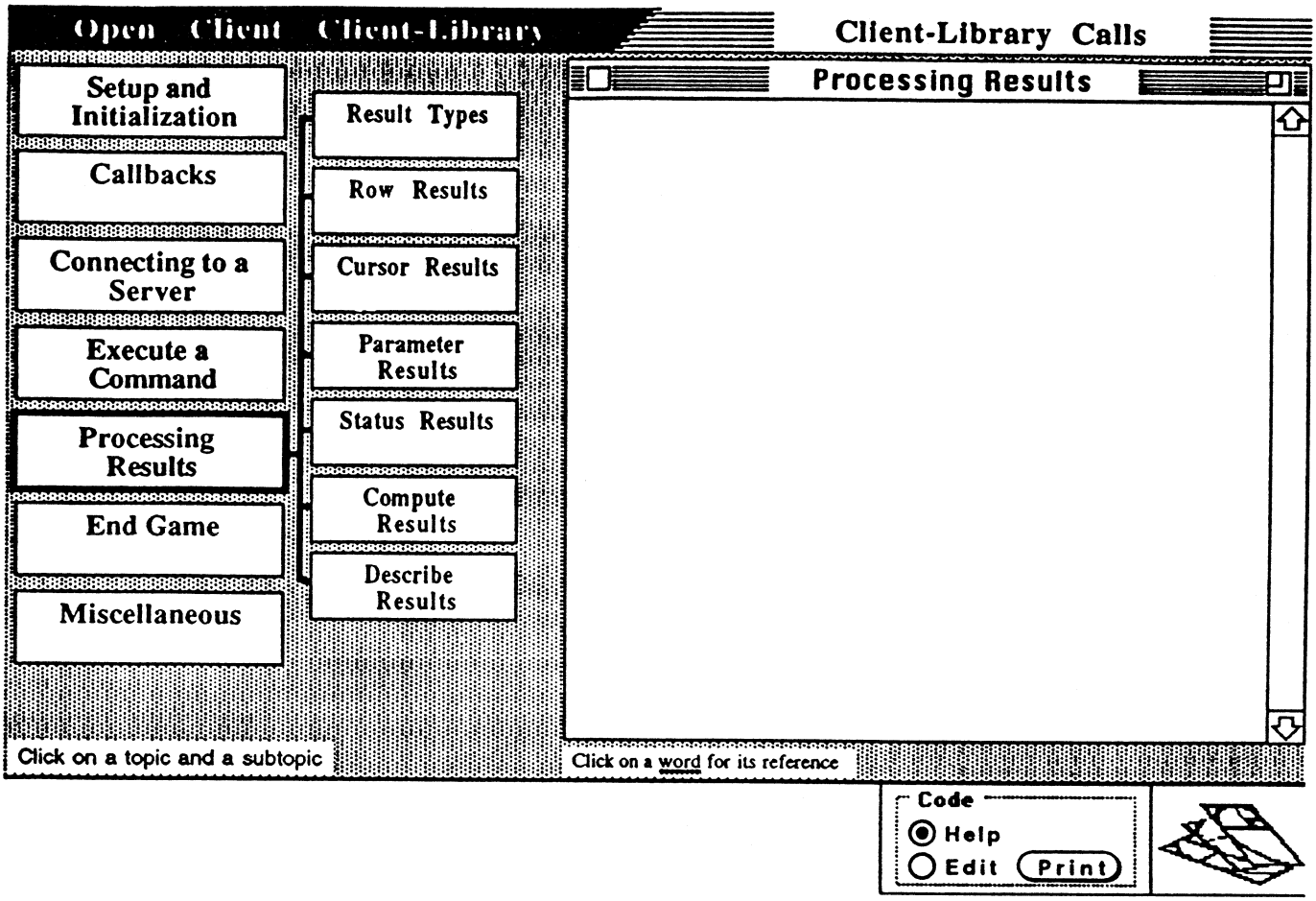
CS_COMMAND *cmd;

Parameters
cmd - A pointer to the CS_COMMAND structure managing a client/ server operation.

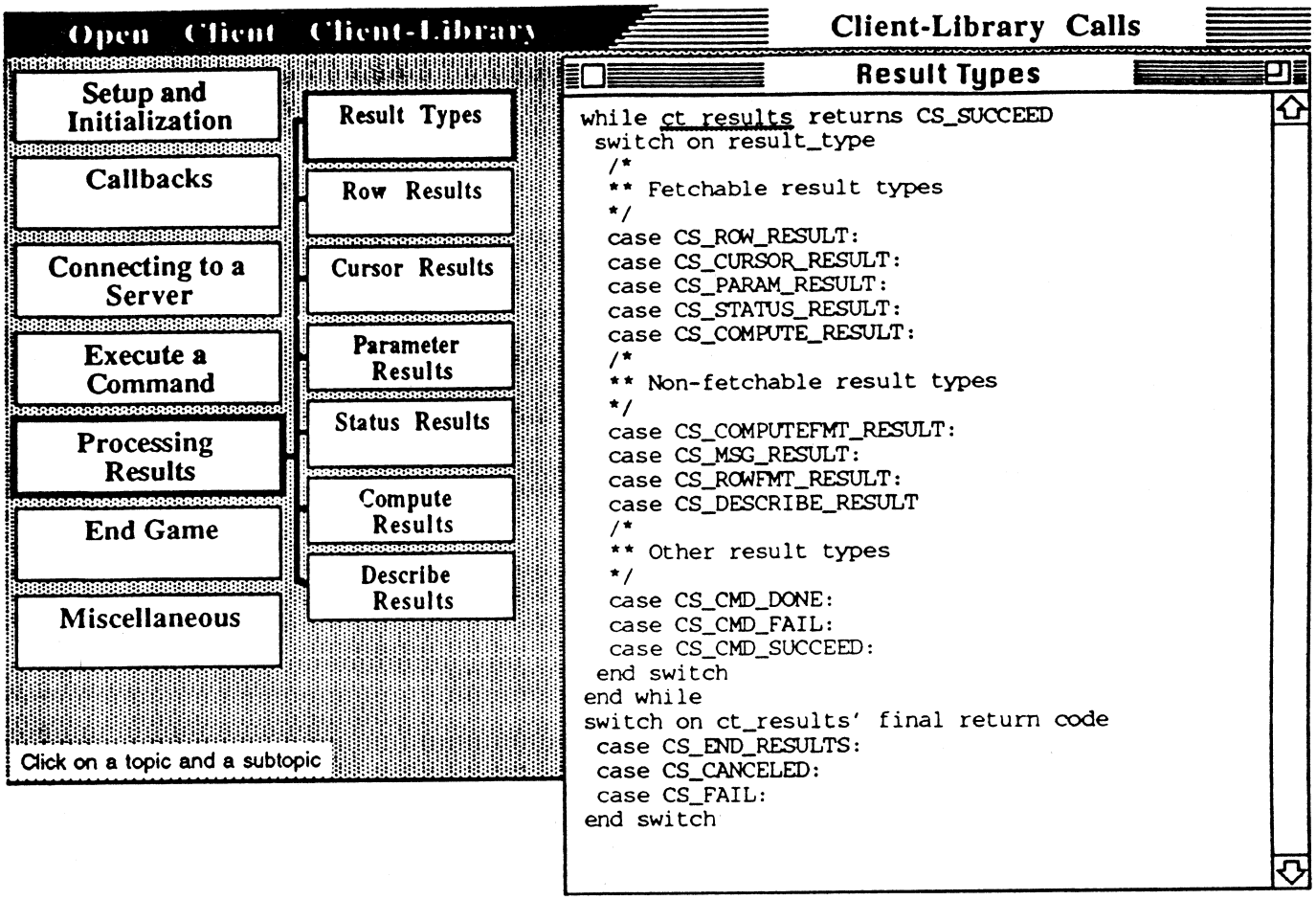
Comments
• ct_send sends a command to the server.
• Sending a command to a server is a three step process. To send a command to a server, an application must:
- Initiate the command by calling ct_command, ct_cursor, or ct_dynamic. These routines set up internal structures that are used in developing a command stream to send to the server.
- Pass parameters for the command. Most applications will pass parameters by calling ct_param once for each parameter that the command requires, but it is also possible to pass parameters for Dynamic SQL commands by using ct_dyndesc.
Not all commands require parameters. For example, a remote procedure call command may or may not require parameters, depending on the stored procedure being called. ct_send does not wait for a response from the server. An application must call ct_results to verify the success of the command and to set up the command results for processing.
- Call ct_send to send the command stream to the server.
• ct_send uses an asynchronous write and not does wait for a response from the server. An application must call ct_results to verify the success of the command and to set up the command results for processing.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed. If ct_send returns CS_FAIL, an application must call ct_cancel with type as CS_CANCEL_ALL before using the affected command structure to send another command.
CS_CANCELED - The routine was canceled.
CS_PENDING - Asynchronous network I/O is in effect.
CS_BUSY - An asynchronous operation is already pending for this connection. For more information, see the Asynchronous Programming topics page.

See Also
ct_command, ct_cursor, ct_dynamic, ct_fetch, ct_param, ct_results



• After an application sends a command to a server, it must process any results generated by the command.



• A single command can generate more than one type of result. For example, a language command that executes a stored procedure can generate multiple regular row and compute row result sets, a parameter result set, and a return status result set. For this reason, it is important that applications be coded to handle all types of results that a server can generate.

• Most synchronous Client-Library programs will process results using a loop controlled by **ct_results**. Inside the loop, a switch takes place on the type of result that is currently available for processing, as indicated by the value of ct_results' parameter result_type. Different types of results require different types of processing.

• result_type is also used to indicate the outcome of a server command that returns no results, for example an insert or delete command returns CS_CMD_SUCCEED.

ct_results

Function
Sets up result data to be processed.

Syntax
CS_RETCODE ct_results(cmd, result_type)

CS_COMMAND *cmd;
CS_INT *result_type;

Parameters
cmd - A pointer to the CS_COMMAND structure managing a client/ server operation.
result_type - A pointer to an integer variable which ct_results sets to indicate the current type of result. result_type is an optional parameter.
The following lists the possible values of *result_type:
Value of *result_type:
CS_CMD_DONE : The results of a logical command have been completelv processed.
CS_CMD_FAIL : The server encountered an error while executing a command.
CS_CMD_SUCCEED : The success of a command that returns no data.
CS_COMPUTE_RESULT : Compute row results.
CS_CURSOR_RESULT : Cursor row results.
CS_PARAM_RESULT : Return parameter results.
CS_ROW_RESULT : Regular row results.
CS_STATUS_RESULT : Stored procedure return status results.
CS_COMPUTEFMT_RESULT : Compute format information.
CS_ROWFMT_RESULT : Row format information.
CS_MSG_RESULT : Message arrival.
CS_DESCRIBE_RESULT : Dynamic SQL descriptive information.

Comments
• ct_results sets up result data for processing. An application calls ct_results after sending a command to the server via ct_send, and before reading the results of that command (if any) via ct_fetch.
• "Result data" is an umbrella term for all the types of data that a server can return to an application. These types of data include:
- Regular rows.
- Cursor rows.
- Return parameters.
- Stored procedure return status numbers.
- Compute rows.
- Dynamic SQL descriptive information.
- Regular row and compute row format information.
- Messages.
ct_results is used to set up all of these types of results for processing.
Don't confuse message results with server error and informational messages. See the Error and Message Handling topics page for a discussion of error and informational messages.
• Result data is returned to an application in the form of "result sets". A result set includes only a single type of result data. For example, a regular row result set contains only regular rows, and a return parameter result set contains only return parameters.

The ct_results Loop
• Because a command can generate multiple result sets, an application must call ct_results as long as it continues to return CS_SUCCEED, indicating that results are available. The simplest way to do this is in a loop that terminates when ct_results fails to return CS_SUCCEED. After the loop, an application can use a case-type statement to test ct_results' final return code to determine why the loop terminated.
• Results are returned to an application in the order in which they are produced. However, this order is not always easy to predict. For example, when an application calls a stored procedure that in turn calls another stored procedure, the application might receive a number of regular row and compute row result sets, as well as a return parameter and a return status result set. The order in which these results are returned depends on how the stored procedures are written.
For this reason, it is recommended that an application's ct_results loop be coded so that control drops into a case-type statement that handles all types of results that can be received. The return parameter result_type indicates symbolically what type of result data the result set contains.

When are the Results of a Command Completely Processed?
• ct_results sets *result_type to CS_CMD_DONE to indicate that the results of a "logical command" have been completely processed. A logical command is defined as any command defined via ct_command, ct_dynamic, or ct_cursor, with the following exceptions:
- Each Transact-SQL select statement inside a stored procedure is a logical command. Other Transact-SQL statements inside stored procedures do not count as logical commands.
- Each Transact-SQL statement executed by a dynamic SQL command is a distinct logical command.
- Each Transact-SQL statement in a language command is a logical command.
For example, suppose a Client-Library language command contains the following Transact-SQL statements:

```
 select type, price
  from titles
  order by type, price
  compute sum(price) by type
 select type, price, advance
  from titles
  order by price, advance
  compute sum(price), max(advance) by type
```

When calling ct_results to process the results of this language command, an application would see the following *result_types:

```
CS_ROW_RESULT      Row and compute results from
CS_COMPUTE_RESULT      the first select,
...      repeated as many times as the
     value of the type column
     changes.
CS_CMD_DONE      Indicates that the results
     of the first query have been
     processed.
CS_ROW_RESULT      Row and compute results from
CS_COMPUTE_RESULT      the second select,
...      repeated as many times as the
     value of the type column
     changes.
CS_CMD_DONE      Indicates that the results of
     the second query have been
     processed.
```

• A *result_type of CS_CMD_SUCCEED or CS_CMD_FAIL is immediately followed by a *result_type of CS_CMD_DONE.
• A connection has pending results if it has not processed all of the results generated by a Client-Library command. Usually, an application cannot send a new command on a connection with pending results. An exception to this rule occurs for CS_CURSOR_RESULT results. For more information on this exception, see ''Connection and Command Rules'' on page 3-5 of the Client-Library Programmer's Guide.

Canceling Results
• To cancel remaining results from a command (and eliminate the need to continue calling ct_results until it fails to return CS_SUCCEED), call ct_cancel.

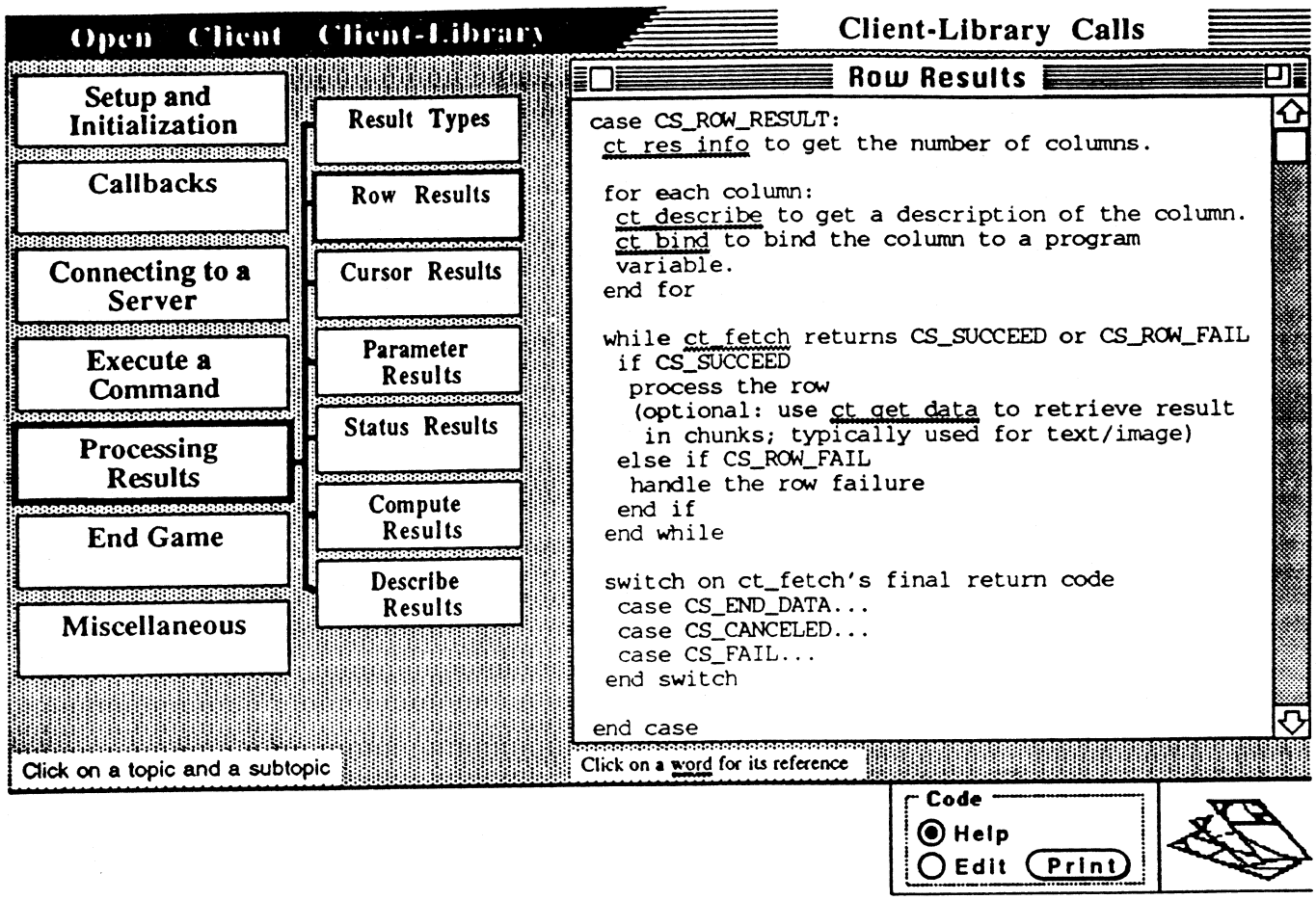
Special Kinds of Result Sets
• A message result set contains no actual result data. Rather, a message has an "id". An application can call ct_res_info to get a message's id. In addition to an id, messages can have parameters. Message parameters are returned to an application as a parameter result set, immediately following the message result set.
• Row format and compute format result sets contains no actual result data. Instead, format result sets contain formatting information for the regular row or compute row result sets with which they are associated.
This type of format information is of use primarily in gateway applications, which need to repackage SQL Server format information before sending it to a foreign client. After ct_results indicates format results, a gateway application can retrieve format information by calling ct_describe, ct_res_info, ct_compute_info, ct_br_column, and ct_br_tables.
All format information for a command is returned before any data. That is, the row format and compute format result sets for a command precede the regular row and compute row result sets generated by the command.
An application will not receive format results unless the Client-Library CS_EXPOSE_FMTS property is set to CS_TRUE.
• A describe result set contains no actual result data. Instead, a describe result set contain descriptive information generated by a Dynamic SQL describe input or describe

output command. After ct_results indicates describe results, an application can retrieve information by calling ct_describe and ct_dyndesc.

ct_results and Stored Procedures
• A run-time error on a language command containing an execute statement will generate a *result_type of CS_CMD_FAIL. A run-time error on a statement inside a stored procedure will not generate a CS_CMD_FAIL, however. For example, if the stored procedure contains an insert statement and the user does not have insert permission on the database table, the insert statement will fail, but ct_results will still return CS_SUCCEED. To check for run-time errors inside stored procedures, examine the procedure's return status number, which is returned as a return status result set immediately following the row and parameter results, if any, from the stored procedure. If the error generates a server message, it is also available to the application.

Returns
CS_SUCCEED - A result set is available for processing.
CS_END_RESULTS - No more result sets are available for processing.
CS_FAIL - The routine failed. If ct_results returns CS_FAIL, an application must call ct_cancel with type as CS_CANCEL_ALL before using the affected command structure to send another command.
CS_CANCELED - Results were canceled.
CS_PENDING - Asynchronous network I/O is in effect.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
ct_bind, ct_command, ct_cursor, ct_describe, ct_dynamic, ct_fetch, ct_send



- A regular row result set is generated by the execution of a Transact-SQL select statement on a server.
- A regular row result set contains zero or more rows of tabular data.
- Processing row results is primarily the activity of **binding** and **fetching**. **Binding** is the process of associating a result item with program data space. **Fetching** is the process of retrieving a data instance of a result item. If binding has been specified for a result item, then fetching causes a data instance of the item to be copied into the program data space.
- Other Client-Library calls, such as **ct_res_info** and **ct_describe**, provide information about the results. This information can aid in implementing, programmatically, the binding and fetching.
- Data items too large to be bound can be retrieved in chunks using ct_get_data. Typically, this is used for large text and image data items.

ct_res_info

Function
Retrieve result set information.

Syntax
CS_RETCODE ct_res_info(cmd, type, buffer, buflen, outlen)

```
CS_COMMAND *cmd;
CS_INT type;
CS_VOID *buffer;
CS_INT buflen;
CS_INT *outlen;
```

Parameters
cmd - A pointer to the CS_COMMAND structure managing a client/ server command.
type - The type of information to return. The table in the Summary of Parameters section lists the symbolic values that are legal for type.
buffer - A pointer to the space in which ct_res_info will place the requested information.
If buflen indicates that *buffer is not large enough to hold the requested information, ct_res_info sets *outlen to the length of the requested information and returns CS_FAIL.
buflen - The length, in bytes, of *buffer.
outlen - A pointer to an integer variable.
ct_res_info sets *outlen to the length, in bytes, of the requested information.
If the requested information is larger than buflen bytes, an application can use the value of *outlen to determine how many bytes are needed to hold the information.

Summary of Parameters
CS_BROWSE_INFO : CS_TRUE if browse-mode information is available; CS_FALSE if browse-mode information is not available.
CS_CMD_NUMBER : The number of the command that generated the current result set.
CS_MSG_ID : An integer representing the id of the message that makes up the current result set.
CS_NUM_COMPUTES : The number of compute clauses in the current command.
CS_NUM_DATA : The number of items in the current result set.
CS_ORDERBY_COLS : The select list id numbers of columns specified in a the order by clause of the current command.
CS_ROW_COUNT : The number of rows affected by the current command.
CS_TRANS_STATE : The current server transaction state.

Comments
• ct_res_info returns information about the current result set or the current command. The current command is defined as the command that generated the current result set.
• A result set is a collection of a single type of result data. Result sets are generated by commands. For more information on result sets, see the ct_results manual page and the Results topics page.

Determining Whether Browse-Mode Information is Available
• To determine whether browse-mode information is available, call ct_res_info with type as CS_BROWSE_INFO.
• If browse-mode information is available, an application can call ct_br_column and ct_br_table to retrieve the information. If browse-mode information is not available, calling ct_br_column or ct_br_table will result in a Client-Library error.
• For more information on browse mode, see the Browse Mode topics page in Chapter 2 of this manual.

Retrieving the Command Number for the Current Result Set
• To determine the number of the command that generated the current result set, call ct_res_info with type as CS_CMD_NUMBER.
• Client-Library keeps track of the command number by counting the number of times ct_results returns CS_CMD_DONE.
An application's first call to ct_results following an ct_send call sets the command number to 1. After this, it is incremented each time ct_results is called after returning CS_CMD_DONE.
• CS_CMD_NUMBER is useful in the following cases:
- To find out which Transact-SQL command within a language command generated the current result set.

- To find out which cursor command, in a batch of cursor commands, generated the current result set.
- To find out which select command in a stored procedure generated the current result set.
• A language command contains a string of Transact-SQL text. This text represents one or more Transact-SQL commands. When used against a language command, "command number" refers to the number of the Transact-SQL command in the language command.
For example, the string:

```
select * from authors
select * from titles
insert newauthors
 select *
 from authors
 where city = "San Francisco"
```

represents three Transact-SQL commands, two of which can generate result sets. In this case, the command number that ct_res_info returns can be from 1 to 3, depending on when ct_res_info is called.
• Inside stored procedures, only select statements cause the command number to be incremented. If a stored procedure contains seven Transact-SQL commands, three of which are selects, the command number that ct_res_info returns can be any integer from 1 to 3, depending on which select generated the current result set.
• ct_cursor is used to define a cursor command. Several cursor commands can be defined before they are sent, as a batch, to a server. When used against a cursor command batch "command number" refers to the number of the cursor command in the command batch.
For example, an application can make the following calls:

```
ct_cursor(...OC_CURSOR_DECLARE...);
ct_cursor(...OC_CURSOR_ROWS...);
ct_cursor(...OC_CURSOR_OPEN...);
ct_send();
```

The command number that ct_res_info returns can be 1, 2, or 3, depending on which cursor command generated the current result type.

Retrieving a Message ID
• To retrieve a message id, call ct_res_info with type as CS_MSG_ID.
• Servers can send messages to client applications. Messages are received in the form of "message result sets." Message result sets contain no fetchable data, but rather have an id number.
• Messages can also have parameters. Message parameters are returned to an application as a parameter result set, immediately following the message result set.

Retrieving the Number of Compute Clauses
• To determine the number of compute clauses in the command that generated the current result set, call ct_res_info with type as CS_NUM_COMPUTES.
• A Transact-SQL select statement can contain compute clauses which generate compute result sets.

Retrieving the Number of Result Data Items
• To determine the number of result data items in the current result set, call ct_res_info with type as CS_NUM_DATA.
• Results sets contain result data items. Row, cursor, and compute result sets contain columns, a parameter result set contains parameters, and a status result set contains a status. The columns, parameters, and status are known as "result data items".
• A message result set does not contain any data items.

Retrieving the Number of Columns in an Order-By Clause
• To determine the number of columns in a Transact-SQL select statement's order by clause, call ct_res_info with type as CS_NUM_ORDERS.
• A Transact-SQL select statement can contain an order by clause, which determines how the rows resulting from the select are ordered on presentation.

Retrieving the Column ID's of Order-By Columns
• To get the select list column id's of order-by columns, call ct_res_info with type as CS_ORDERBY_COLS.
• Columns named in an order by clause must also be named in the select list of the select statement. Columns in a select list have a "select list id," which is the number in which they appear in the list. For example, in the following query, au_lname and au_fname have select list id's of 1 and 2 respectively:

```
select au_lname, au_fname from authors
 order by au_fname, au_lname
```

• Given the preceding query, the call:
```
ct_res_info(cmd, CS_ORDERBY_COLS, myspace, 8,
  outlength)
```
sets *myspace to an array of two CS_INTs containing the integers 2 and 1.

Retrieving the Number of Rows for the Current Command
• To determine the number of rows affected by the current command, call ct_res_info with type as CS_ROW_COUNT.
• An application can retrieve a row count after ct_results sets its *result_type parameter to CS_CMD_SUCCEED, CS_CMD_DONE, or CS_CMD_FAIL. A row count is guaranteed to be accurate if ct_results has just set *result_type to CS_CMD_DONE.
• If the command is one that executes a stored procedure, for example a Transact-SQL exec language command or a remote procedure call command, ct_res_info sets *buffer to the number of rows affected by the last statement in the stored procedure that affects rows.
• ct_res_info sets *buffer to CS_NO_COUNT if any of the following are true:
- The Transact-SQL command fails for any reason, such as a syntax error.
- The command is one that never affects rows, such as a Transact-SQL print command.
- The command executes a stored procedure that does not affect any rows.
- The CS_OPT_NOCOUNT option is on.

Retrieving the Current Server Transaction State
• To determine the current server transaction state, call ct_res_info with type as CS_TRANS_STATE.
• For more information about server transaction states, see ''Server Transaction States'' on page 2-56.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed. ct_res_info returns CS_FAIL if the requested information is larger than buflen bytes, or if there is no current result set.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
ct_cmd_props, ct_con_props, ct_results, Options

ct_describe

Function
Retreive a description of result data.

Syntax
CS_RETCODE ct_describe(cmd, item, datafmt)

CS_COMMAND *cmd;
CS_INT item;
CS_DATAFMT *datafmt;

Parameters
cmd - A pointer to the CS_COMMAND structure managing a client/ server operation.
item - An integer representing the result item of interest.
datafmt - A pointer to a CS_DATAFMT structure. ct_describe fills *datafmt with a description of the result data item referenced by item. ct_describe fills in the following fields in the CS_DATAFMT:

Comments
• ct_describe returns a complete description of a result data item in the current result set. Result data items include regular result columns, return parameters, stored procedure return status numbers, and compute result columns.
An application can also use ct_describe to retrieve format information. Client-Library indicates that format information is available by setting ct_results' *result_type to CS_ROWFMT_RESULT or CS_COMPUTEFMT_RESULT.
• An application can call ct_res_info to find out how many result items are present in the current result set.
• An application generally needs to call ct_describe to describe a result data item before it binds the result item to a program variable using ct_bind.
• See the CS_DATAFMT topics page for a description of the CS_DATAFMT structure.
• See the Results topics page for a description of result types.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed. ct_describe returns CS_FAIL if item does not represent a valid result data item.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
ct_bind, ct_fetch, ct_res_info, ct_results, Results

ct_bind

Function
Bind server results to program variables.

Syntax
CS_RETCODE ct_bind(cmd, item, datafmt, buffer, copied, indicator)

CS_COMMAND *cmd;
CS_INT item;
CS_DATAFMT *datafmt;
CS_VOID *buffer;
CS_INT *copied;
CS_SMALLINT *indicator;

Parameters
cmd - A pointer to the CS_COMMAND structure managing a client/ server operation.
item - An integer representing the number of the column, parameter, or status to bind. To clear all bindings, pass item as CS_UNUSED, with datafmt, buffer, copied, and indicator as NULL.
datafmt - A pointer to the CS_DATAFMT structure that describes the destination variable(s).
buffer - The address of an array of datafmt->count variables, each of which is of size datafmt->maxlength. *buffer is the program variable or variables to which ct_bind binds the server results. When the application calls ct_fetch to fetch the result data, it is copied into this space.
If buffer is NULL, ct_bind clears the binding for this result item. Note that if buffer is NULL, datafmt, copied, and indicator must also be NULL.
copied - The address of an array of datafmtÆcount integers. At fetch time, ct_fetch fills this array with the lengths of the copied data. copied is an optional parameter and can be passed as NULL.
indicator - The address of an array of datafmtÆcount integer variables. At fetch time, each variable is used to indicate certain conditions about the fetched data. The following lists the values that an indicator variable can have:
-1 : The fetched data was NULL. In this case, no data is copied to *buffer.
0 : The fetch was successful.
integer value : The actual length of the server data, if the fetch resulted in truncation.

Comments
• ct_bind binds server results to program variables.
• ct_bind can be used to bind a regular or cursor result column, a compute column, a return parameter, or a stored procedure status number. When binding a regular or cursor column, multiple rows of the column can be bound with a single call to ct_bind. Message, describe, row format, and compute format results are not bound. This is because result sets of type CS_MSG_RESULT, CS_DESCRIBE_RESULT, CS_ROWFMT_RESULT, and CS_COMPUTEFMT_RESULT contain no fetchable data. Instead, these result sets indicate that certain types of information are available. An application can retrieve the information by calling other Client-Library routines, such as ct_res_info. For more information on how to process results, see the Results topics page in Chapter 2 of this manual.
• Binding associates a result data item with a program variable. At fetch time, each ct_fetch call copies a row-instance of the data item into the variable with which the item is associated. If a result data item is very large (for example, a large text or image column), it is often more convenient for an application to use ct_get_data to retrieve the data item's value in chunks, rather than copying the entire value to a bound variable. For more information on ct_get_data, see the ct_get_data manual page in this chapter, and the Text and Image topics page in Chapter 2.
• ct_bind binds only the current result type. ct_results indicates the current result type via its result_type parameter. For example, if ct_results sets *result_type to CS_STATUS_RESULT, a return status is available for binding.
• An application can call ct_res_info to determine the number of items in the current result set, and can call ct_describe to get a description of each item.
• An application can only bind a result item to a single program variable. If an application binds a result item to multiple variables, only the last binding takes effect.
• An application can re-bind while actively fetching rows. That is, an application can call ct_bind inside a ct_fetch loop if it needs to change a result item's binding.
• Binding for a particular type of result remains in effect until ct_results returns CS_CMD_DONE to indicate that the results of a logical command are completely processed. This saves an application the trouble of rebinding interspersed regular row and compute

row result sets that are generated by the same command.
For example, a language command containing a select statement with compute and order by clauses can generate multiple regular row result sets intermixed with compute row result sets. Because they are generated by the same command, each regular row result set and each compute row result set will contain identical columns. An application need only bind the first regular row result set and the first compute result set. These bindings will remain in effect until all of the results of the command are processed.
• An application can use ct_bind to bind to Open Client user-defined datatypes for which conversion routines have been installed. To install a conversion routine for a user-defined datatype, an application calls cs_set_convert. For more information on Open Client user-defined types, see the Types topics page.

Clearing Bindings
• To clear the binding for a result item, call ct_bind with buffer, datafmt, copied, and indicator as NULL.
• To clear all bindings, call ct_bind with item as CS_UNUSED and buffer, datafmt, copied, and indicator as NULL.
• It is not an error to clear a non-existent binding.

Array Binding
• Array binding is the process of binding a result column to an array of program variables. At fetch time, multiple rows' worth of the column are copied to the array of variables with a single ct_fetch call. An application indicates array binding by setting datafmtëëëëëÆcount to a value greater than 1.
• Array binding is only practical for regular row and cursor results. This is because other types of results are considered to be the equivalent of a single row.
• When binding columns to arrays, all ct_bind calls in the sequence of calls binding the columns must use the same value for datafmtÆcount. For example, when binding three columns to arrays, it is an error to use a count of five in your first two ct_bind calls and a count of three in the last.
However, an application can intermix counts of 0 and 1. counts of 0 and 1 are considered to be equivalent because they both cause ct_fetch to fetch a single row.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.
CS_BUSY - An asynchronous operation is already pending for this connection.

Common reasons for a ct_bind failure include:
• An illegal datatype specified via datafmt->datatype.
• A bad datafmt->locale pointer.

See Also
ct_describe, ct_fetch, ct_res_info, ct_results, Types

ct_fetch

Function
Fetch result data.

Syntax
CS_RETCODE ct_fetch(cmd, type, offset, option, rows_read)

CS_COMMAND *cmd;
CS_INT type;
CS_INT offset;
CS_INT option;
CS_INT *rows_read;

Parameters
cmd - A pointer to the CS_COMMAND structure managing a client/ server operation.
type - This parameter is currently unused and should be passed as CS_UNUSED in order to ensure compatibility with future versions of Client-Library.
offset -This parameter is currently unused and should be passed as CS_UNUSED in order to ensure compatibility with future versions of Client-Library.
option - This parameter is currently unused and should be passed as CS_UNUSED in order to ensure compatibility with future versions of Client-Library.
rows_read - A pointer to an integer variable. ct_fetch sets rows_read to the number of rows read by the ct_fetch call.
rows_read is an optional parameter intended for use by applications using array binding.

Comments
• ct_fetch fetches result data. "Result data" is an umbrella term for all the types of data that a server can return to an application. These types of data include:
- Regular rows.
- Cursor rows.
- Return parameters, including both message parameters and stored procedure return parameters.
- Stored procedure status numbers.
- Compute rows.
ct_fetch is used to fetch all of these types of data.
• Conceptually, result data is returned to an application in the form of one or more rows that make up a "result set".
Regular row and cursor row result sets can contain more than one row. For example, a regular row result set might contain a hundred rows. If array binding has been specified for the data items in a regular row or cursor row result set, then multiple rows can be fetched with a single call to ct_fetch.
Return parameter, status number, and compute row result sets, however, only contain a single "row." For this reason, even if array binding is specified, only a single row of data is fetched.
• ct_results sets *result_type to indicate the type of result available. ct_results must indicate a result type of CS_ROW_RESULT, CS_CURSOR_RESULT, CS_PARAM_RESULT, CS_STATUS_RESULT, or CS_COMPUTE_RESULT before an application calls ct_fetch.
• After calling ct_results, an application can:
- Process the result set by binding the result items and fetching the data. (A typical application will call ct_describe to get data descriptions, ct_bind to bind result items, ct_fetch to fetch result rows, and ct_get_data, if the result set contains large text or image values. However, an application can also use ct_fetch and ct_dyndesc to process a result set.)
- Discard the result set, using ct_cancel.
• If an application does not cancel a result set, it must completely process the result set by calling ct_fetch as long as ct_fetch continues to indicates that rows are available.
The simplest way to this is in a loop that terminates when ct_fetch fails to return either CS_SUCCEED or CS_ROW_FAIL. After the loop terminates, an application can use a switch-type statement against ct_fetch's final return code to find out what caused the termination.
• If a conversion error occurs when retrieving a result item, the rest of the items in the row are retrieved. If truncation occurs, the indicator variable, if any, provided in the application's ct_bind call for this item is set to the actual length of the result data.

Fetching Regular Rows and Cursor Rows
• Regular rows and cursor rows can be fetched from the server one row at a time, or several rows at once.

• An application indicates the number of rows to be fetched per ct_fetch call via the datafmt->count field in its ct_bind calls that bind result columns to program variables. If datafmt->count is 0 or 1, each call to ct_fetch fetches one row. If datafmt->count is greater than one, then array binding is considered to be in effect and each call to ct_fetch fetches datafmt->count rows. (Note that datafmt->count must have the same value for all ct_bind calls for a result set.)
• When fetching multiple rows, if a conversion error occurs on one of the rows, no more rows are retrieved by this ct_fetch call.

Fetching Return Parameters
• A return parameter result set contains either stored procedure return parameters or message parameters.
• A return parameter result set consists of a single row with a number of columns equal to the number of return parameters.

Fetching a Return Status
• A stored procedure return status result set consists of a single row with a single column, the status number.

Fetching Compute Rows
• Compute rows result from the compute clause of a select statement.
• A compute row result set consists of a single row with a number of columns equal to the number of aggregate operators in the compute clause that generated the row.
• Each compute row is considered to be a distinct result set.

Returns
CS_SUCCEED - The routine completed successfully.
CS_END_DATA - No more rows are available in this result set.
CS_ROW_FAIL - A recoverable error occurred while fetching a row. Recoverable errors include memory allocation failures and conversion errors that occur while copying row values to program variables. An application can continue calling ct_fetch to keep retrieving rows, or can call ct_cancel to cancel the remaining results. ct_fetch places the number of rows fetched before the error occurred in *rows_read.
CS_FAIL - The routine failed. ct_fetch places the number of rows fetched in *rows_read. This number includes the failed row.
CS_CANCELED - The operation was canceled. ct_fetch places the number of rows fetched before the cancel occurred in *rows_read.
CS_PENDING - Asynchronous network I/O is in effect.
CS_BUSY - An asynchronous operation is already pending for this connection.

A common reason for a ct_fetch failure is that a program variable specified via ct_bind is not large enough for a fetched data item.

See Also
ct_bind, ct_describe, ct_results

ct_get_data

Function
Read a chunk of data from the server.

Syntax
CS_RETCODE ct_get_data(cmd, item, buffer, buflen, outlen)

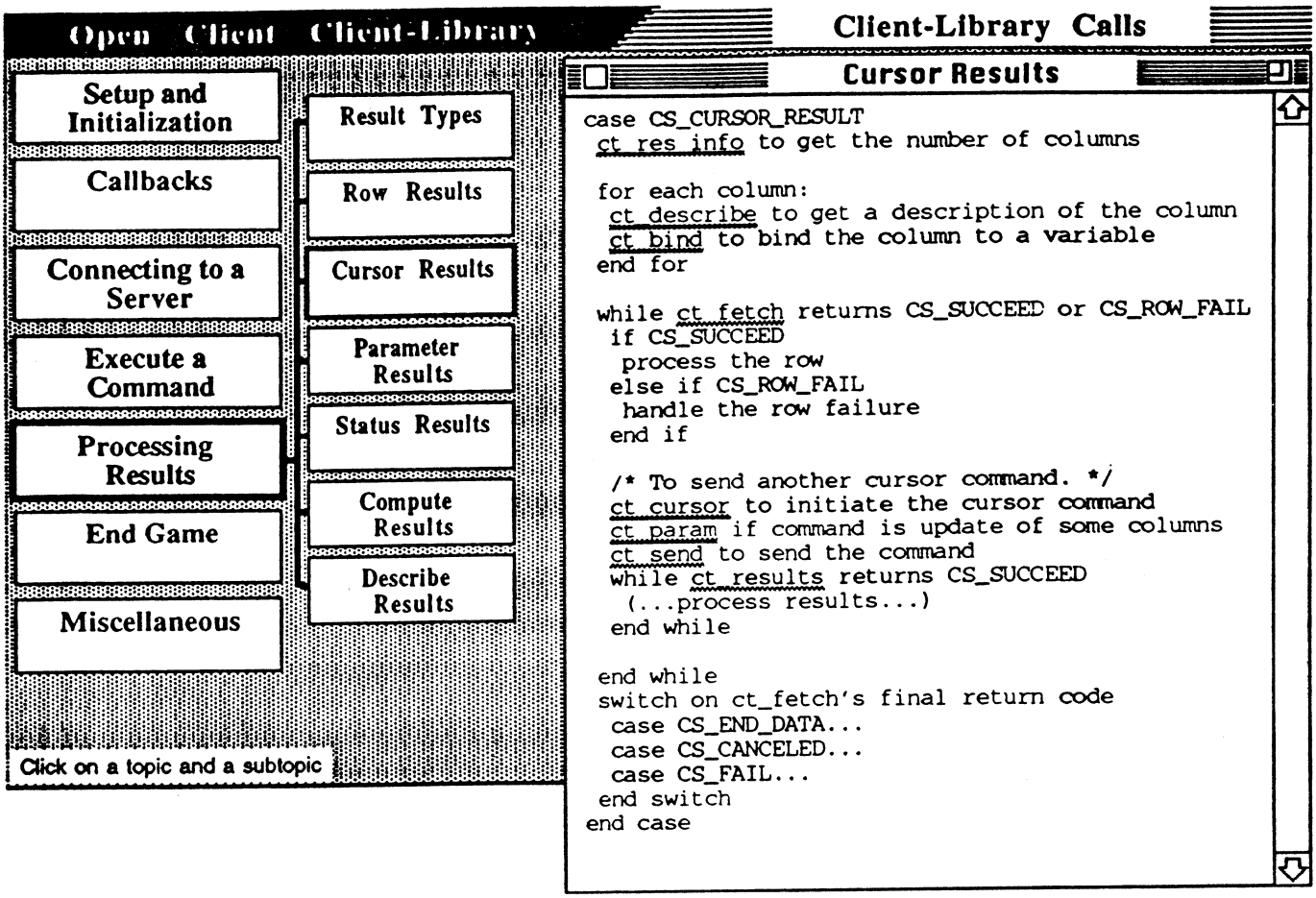
```
CS_COMMAND *cmd;
CS_INT item;
CS_VOID *buffer;
CS_INT buflen;
CS_INT *outlen;
```

Parameters
cmd - A pointer to the CS_COMMAND structure managing a client/ server operation.
item - An integer representing the data item of interest. When using ct_get_data to retrieve data for more than one item in a result set, item can be incremented only; that is, an application cannot retrieve data for item number 3 after it has retrieved data for item number 4.
buffer - A pointer to data space. ct_get_data fills *buffer with a buflen-sized chunk of the column's value.
buflen - The length, in bytes, of *buffer.
outlen - A pointer to an integer variable. ct_get_data sets outlen to the number of bytes placed in *buffer.

Comments
• ct_get_data reads a chunk of data from the server.
• ct_get_data is typically called in a loop retrieve large text or image values, although it can be used on columns of any datatype. Each call to ct_get_data retrieves a buflen-sized chunk of data.
• ct_get_data retrieves data exactly as it is sent by the server. No conversion is performed. For this reason, care must be taken when interpreting data contained in *buffer. In particular, CS_CHAR data may not be null-terminated and multi-byte character strings may be broken within a byte sequence defining a single character.
• An application calls ct_get_data after calling ct_fetch to fetch the row of interest. If array binding was indicated in an earlier call to ct_bind, the application cannot use ct_get_data.
• Only those columns following the last bound column are available to ct_get_data. Data in unbound columns that precede bound columns is discarded.
• Once data has been retrieved for a column, it is no longer available.
• If an application reads a text or image column that it will need to update at a later time, it needs to retrieve an I/O descriptor for the column. To do this, an application can call ct_data_info after calling ct_get_data for the column.
• For more information on how to use ct_get_data, see the Text and Image topics page.

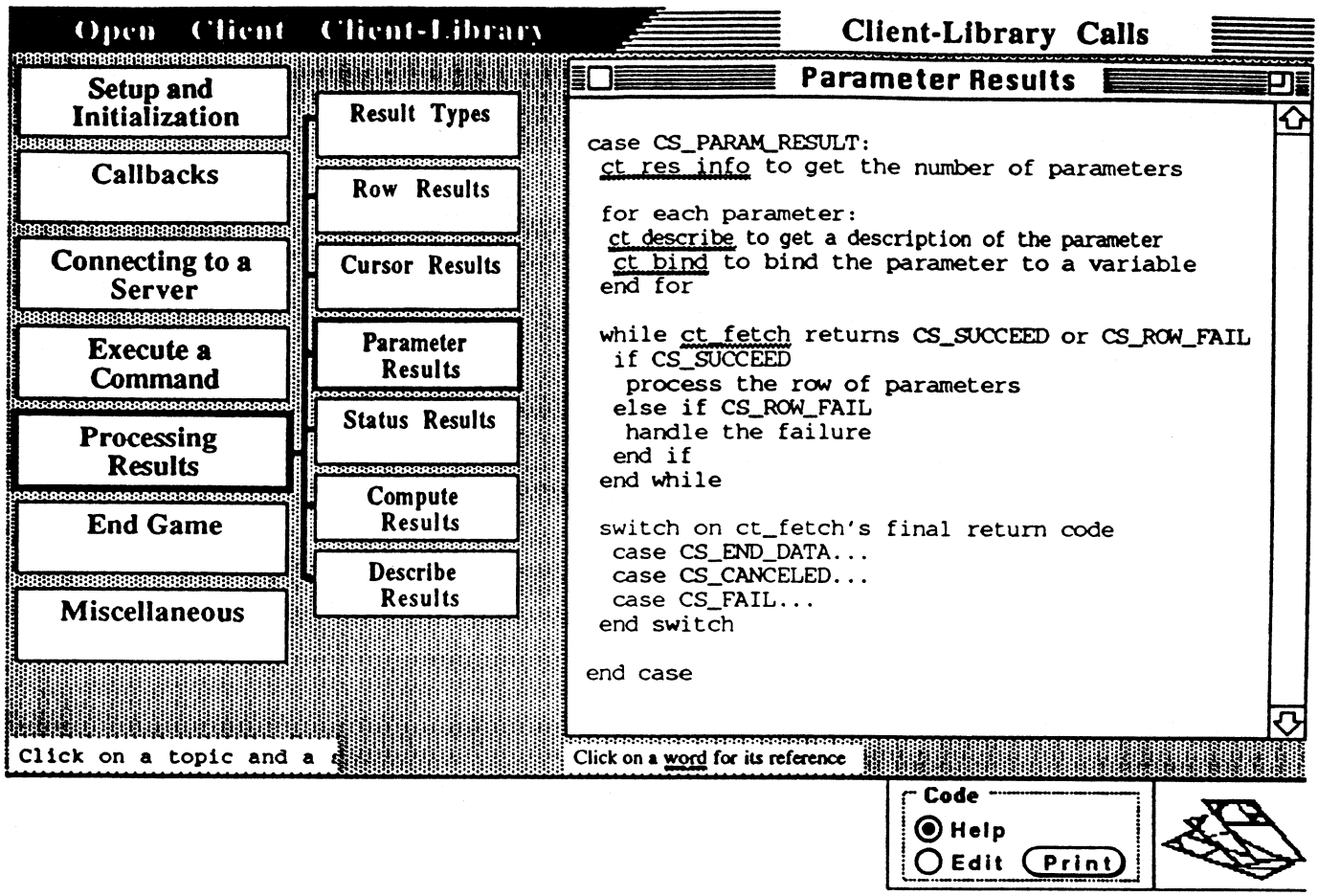
Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.
CS_END_ITEM - The end of this item's value. More items are available for this row.
CS_END_DATA - The end of this item's value. No more data items are available for this row.
CS_CANCELED - The get data operation was canceled.
CS_PENDING - Asynchronous network I/O is in effect.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
ct_data_info, ct_send_data, Text and Image



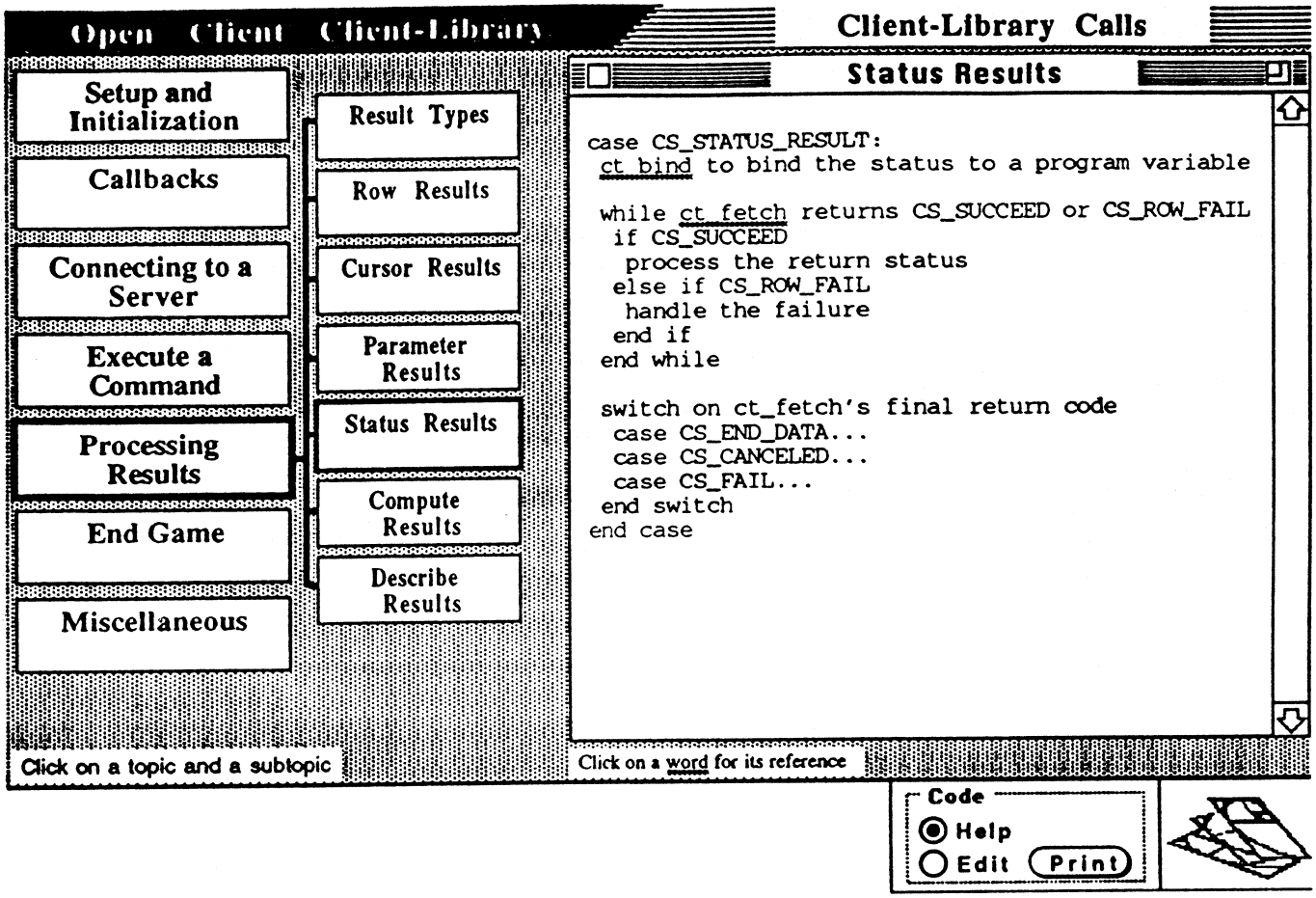
- A cursor row result set is generated when a cursor open command is executed.

- In general, when a synchronous application sends a command to a server, it cannot send another command on the same command structure until **ct_results** indicates that the results of the first command have been completely processed.
An exception to this rule occurs when **ct_results** indicates cursor results. In this case, an application can call **ct_cursor** and **ct_send** to send cursor commands at the same time it is processing the cursor result set. A cursor update or delete command can be done with the command handle currently being processed. Any other cursor command would need a different command handle than the one currently being processed, however it can be associated with the same connection.



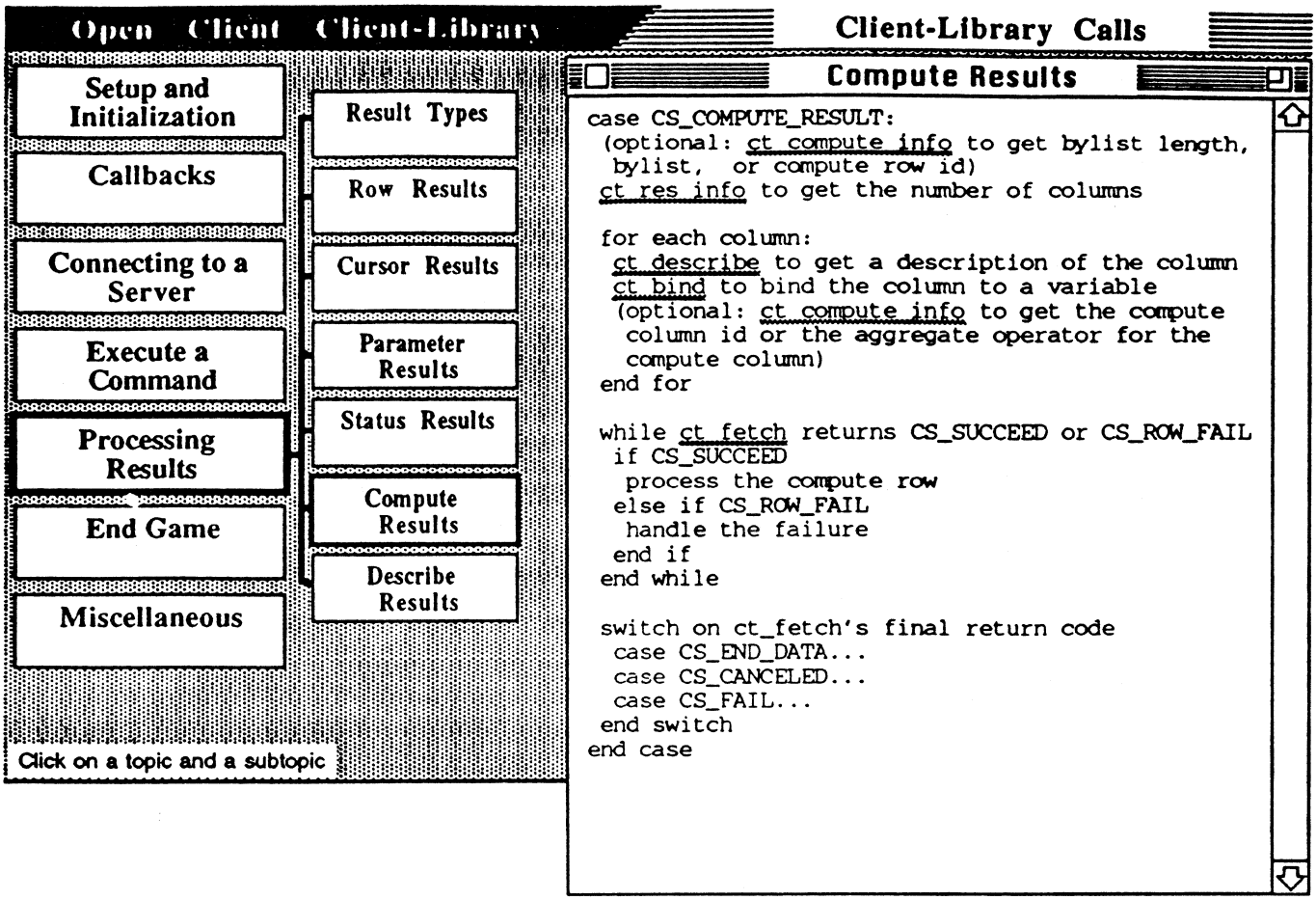
¥ A parameter result set contains one or more rows of parameters.

¥ There are two types of parameters: **message parameters** and **stored return parameters**.



- A status result set is generated by the execution of a stored procedure. All stored procedures that run on a SQL Server version 4.0 or greater return a status number.

- A status result set consists of a single row containing a return status.



- A compute result set is generated by the execution of a Transact-SQL select statement that contains a compute clause.

- A compute row result set contains a single row of tabular data with a number of columns equal to the number of columns listed in the compute clause that generated the compute row.

- In addition to **ct_res_info**, **ct_describe**, **ct_bind**, and **ct_fetch**, an application may call **ct_compute_info** while processing compute row results. **ct_compute_info** provides a variety of compute row information, including the compute id of the compute row and the operator types for columns in the row.

ct_compute_info

Function
Retrieve compute result information.

Syntax
CS_RETCODE ct_compute_info(cmd, type, colnum, buffer, buflen, outlen)

CS_COMMAND *cmd;
CS_INT type;
CS_INT colnum;
CS_VOID *buffer;
CS_INT buflen;
CS_INT *outlen;

Parameters
cmd - A pointer to the CS_COMMAND structure managing a client/ server command.
type - The type of information to return. For a list of the symbolic values that are legal for type, see the chart in the Summary of Parameters section.
colnum - The number of the compute column of interest, as it appears in the compute row result set. Compute columns appear in the order in which they are listed in the compute clause of a select statement. The first column is number 1, the second number 2, and so forth.
buffer - A pointer to the space in which ct_compute_info will place the requested information. If buflen indicates that *buffer is not large enough to hold the requested information, ct_compute_info returns CS_FAIL.
buflen - The length, in bytes, of the *buffer data space.
outlen - A pointer to an integer variable. When type is CS_COMP_BYLIST, ct_compute_info sets *outlen to the number of elements in the bylist array. When type has any other value, ct_compute_info sets *outlen to the length, in bytes, of the requested information. If the requested information is larger than buflen bytes, an application can use the value of *outlen to determine how many bytes are needed to hold the information.

Comments
• ct_compute_info returns compute result information.
• Compute rows result from the compute clause of a select statement. A compute clause generates a compute row every time the value of its by column-list changes. A compute row will contain one column for each aggregate operator in the compute clause. If a select statement contains multiple compute clauses, separate compute rows are generated by each clause. Each compute row returned by the server is considered to be a distinct result set. That is, each result set of type CS_COMPUTE_RESULT will contain exactly one row.
• It is legal to call ct_compute_info when compute information is available; that is, after ct_results returns CS_COMPUTE_RESULT or CS_COMPUTEFMT.
• Each section below contains information about a particular type of compute result information.

The Bylist for a Compute Row
• A select statement's compute clause may contain the keyword by, followed by a list of columns. This list, known as the "bylist," divides the results into subgroups, based on changing values in the specified columns. The compute clause's aggregate operators are applied to each subgroup, generating a compute row for each subgroup.

The Select-List Column ID for a Compute Column
• The select-list column id for a compute column is the select-list id of the column from which the compute column derives.

The Compute ID for this Compute Row
• A SQL select statement can have multiple compute clauses, each of which returns a separate compute row. The compute id corresponding to the first compute clause in a select statement is 1.

The Aggregate Operator for a Particular Compute Row Column
• When called with type as CS_COMP_OP, ct_compute_info sets *buffer to one of the following aggregate operator types:
CS_OP_AVG : Average aggregate operator.
CS_OP_COUNT : Count aggregate operator.
CS_OP_MAX : Maximum aggregate operator.
CS_OP_MIN : Minimum aggregate operator.

CS_OP_SUM : Sum aggregate operator.

Examples

Assume the following command has been executed:

```
select dept, name, year, sales from employee
  order by dept, name, year
  compute count(name) by dept, name
```

1. The call:

ct_compute_info(cmd, CS_BYLIST_LEN, CS_UNUSED, mybuffer, CS_UNUSED, CS_UNUSED)
sets *mybuffer to 2, because there are two items in the bylist.

2. The call:

ct_compute_info(cmd, CS_COMP_BYLIST, CS_UNUSED,mybuffer, 20, outlength)
copies the CS_SMALLINT values 1 and 2 into *mybuffer to indicate that the bylist is composed of columns 1 and 2 from the select list.

3. The call:

ct_compute_info(cmd, CS_COMP_COLID, 1, mybuffer,CS_UNUSED,NULL)
sets *mybuffer to 2, since name is the second column in the select list.

4. The call:

ct_compute_info(cmd, CS_COMP_ID, CS_UNUSED, mybuffer, CS_UNUSED, NULL)
sets *mybuffer to 1 because there is only a single compute clause in the select statement.

5. The call:

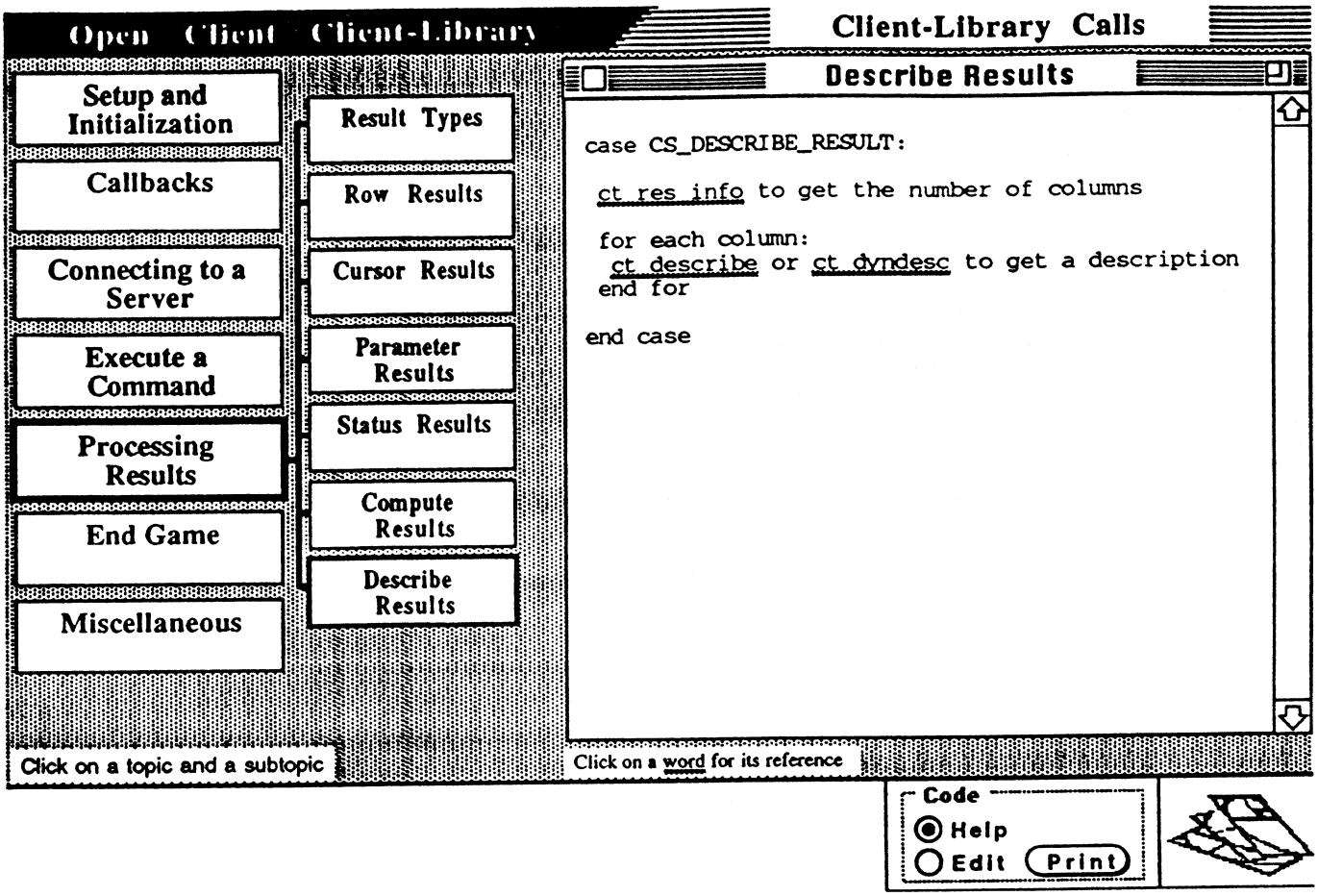
ct_compute_info(cmd, CS_COMP_OP, 1, mybuffer, CS_UNUSED, NULL)
sets *mybuffer to the symbolic value CS_OP_COUNT, since the aggregate operator for the first compute column is a count.

Returns

CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also

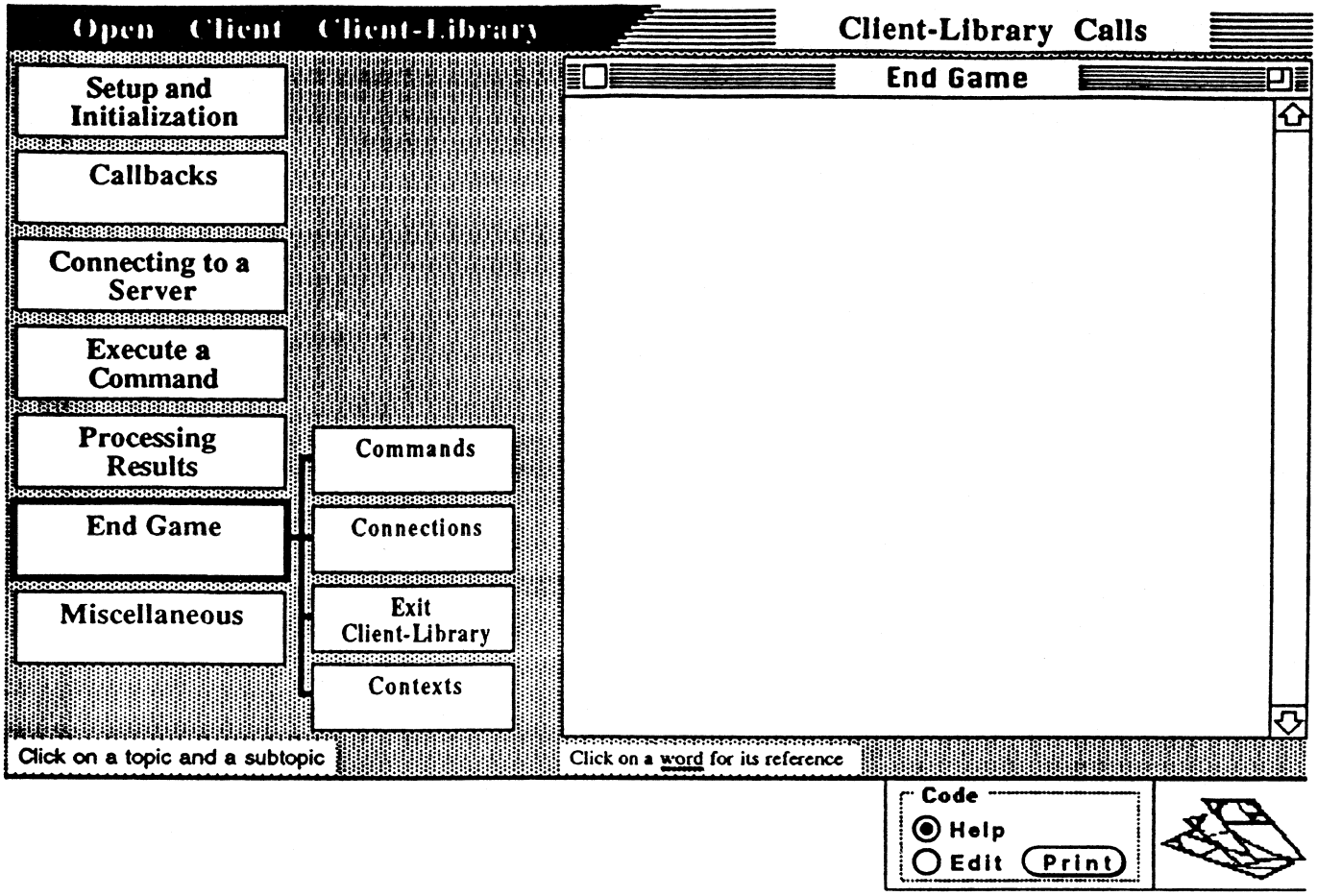
ct_bind, ct_describe, ct_res_info, ct_results



• A describe result set does not contain fetchable data. It indicates the existence of descriptive information returned as the result of a Dynamic SQL describe input or describe output command.

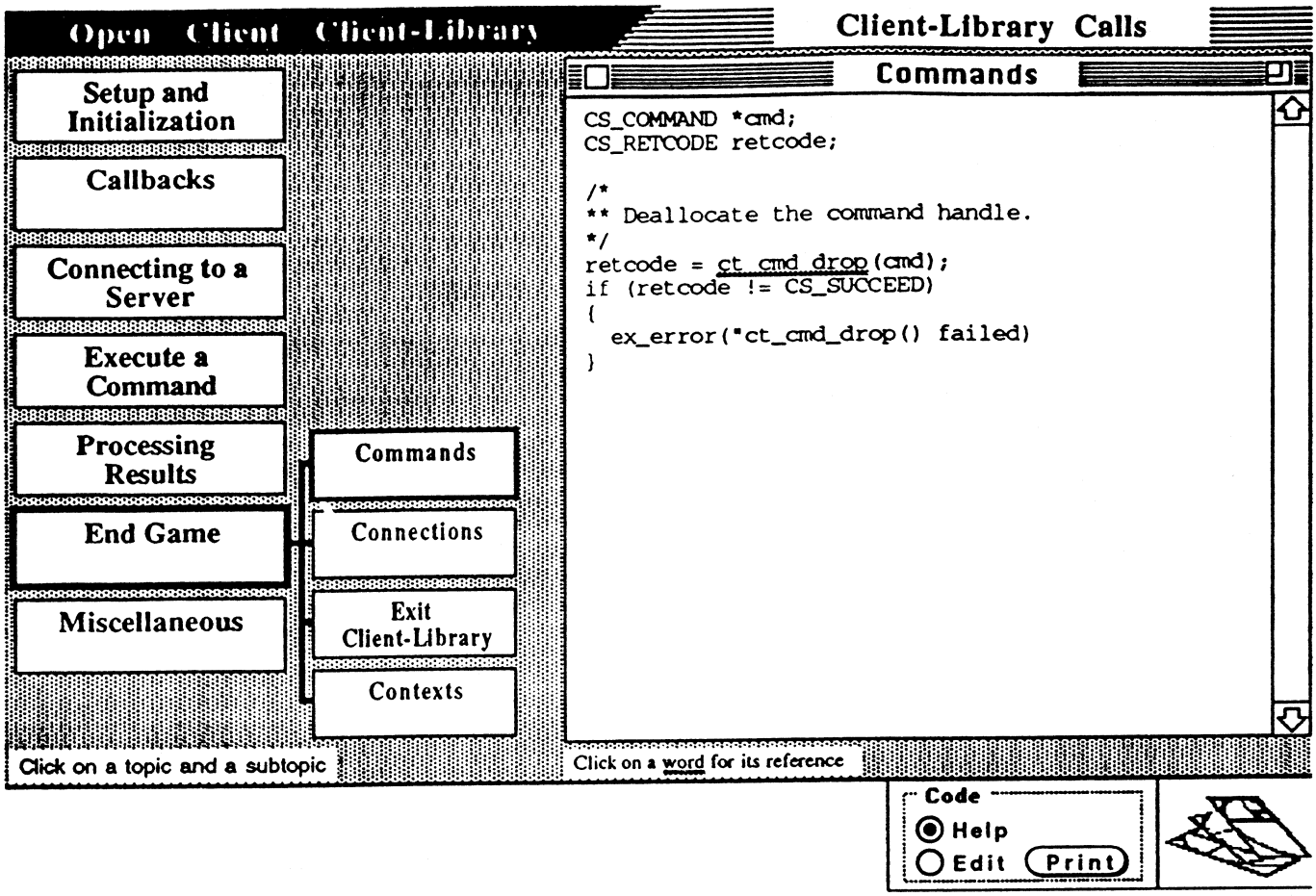
• An application can retrieve this descriptive information by calling **ct_describe** or **ct_dyndesc**.

• For more information on Dynamic SQL, see the Dynamic SQL topics page in Chapter2 of the Client-Library Reference Manual.



- Before exiting, a Client-Library application must:

1) De-allocate all command structures for each connection.
2) Close and de-allocate all open connections.
3) Exit Client-Library.
4) De-allocate all context structures.



• An application calls **ct_cmd_drop** to de-allocate a command structure. It is an error to de-allocate a command structure which has pending results or an open cursor.

ct_cmd_drop

Function
De-allocate a CS_COMMAND structure.

Syntax
CS_RETCODE ct_cmd_drop(cmd)

CS_COMMAND *cmd;

Parameters
cmd - A pointer to a CS_COMMAND structure.

Comments
- ct_cmd_drop de-allocates a CS_COMMAND structure.
- A CS_COMMAND structure is a control structure that a Client-Library application uses to send commands to a server and process the results of those commands.
- Once a command structure has been de-allocated, it cannot be re-used. To allocate a new CS_COMMAND structure, an application can call ct_cmd_alloc.
- ct_cmd_drop returns CS_FAIL if *cmd is managing a cursor or if it has any results pending. Before de-allocating a command structure, an application should process or cancel any pending results and close and de-allocate any open cursors.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed. ct_cmd_drop returns CS_FAIL if *cmd is managing a cursor or if it has any results pending.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
ct_command, ct_cmd_alloc

ct_close

Function
Close a server connection.

Syntax
CS_RETCODE ct_close(connection, option)

CS_CONNECTION *connection;
CS_INT option;

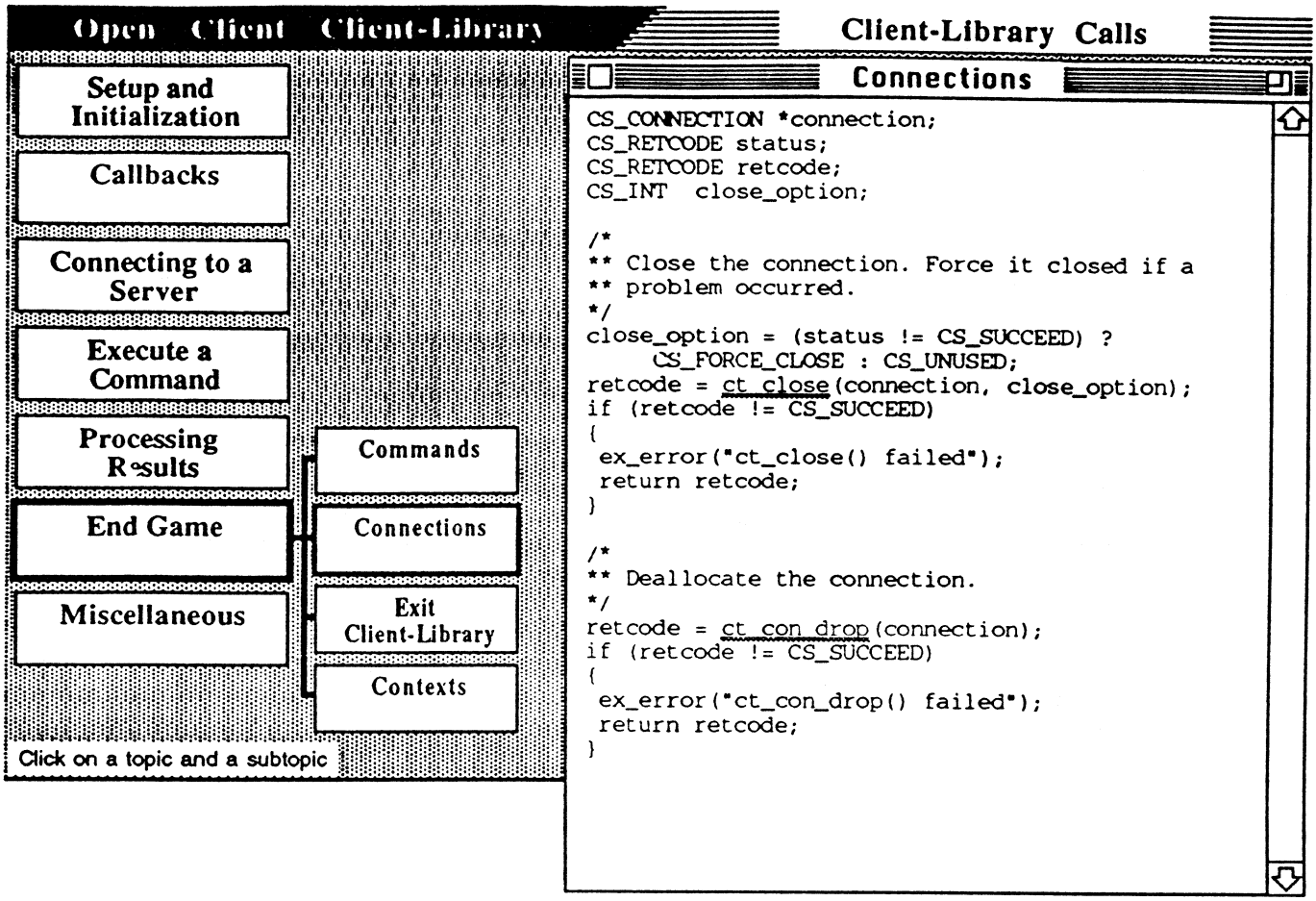
Parameters
connection - A pointer to a CS_CONNECTION structure. A CS_CONNECTION structure contains information about a particular client/ server connection.
option - The option, if any, to use for the close. The following lists the symbolic values that are legal for option:
CS_UNUSED : (10.0+ servers only) Default behavior. ct_close sends a logout message to the server and reads the response to this message before closing the connection.
If the connection has results pending, ct_close returns CS_FAIL.
CS_FORCE_CLOSE : The connection is closed whether or not results are pending, and without notifying the server. This option is primarily for use when an application is hung waiting for a server response. It is also useful if ct_results, ct_fetch, or ct_cancel returns CS_FAIL.

Comments
• ct_close closes a server connection.
• To de-allocate a CS_CONNECTION, an application can call ct_con_drop after the connection has been successfully closed.
• If the connection is using asynchronous network I/O, ct_close returns CS_PENDING. When the server response arrives, Client-Library closes the connection and then calls the completion callback installed for the connection, if any.
• When a connection is closed, all open cursors on that connection are automatically closed.
• The behavior of ct_close depends on the value of option, which determines the type of close. Each section below contains information on a type of close:

Default Close Behavior
• If the connection has any pending results, ct_close returns CS_FAIL. If the connection has an open cursor, the server closes the cursor when Client-Library closes the connection.
• Before terminating the connection with the server, ct_close sends a logout message to the server and reads the response to this message. The contents of this message do not affect ct_close's behavior.
• An application cannot call ct_close(CS_UNUSED) when an asynchronous operation is pending.

CS_FORCE_CLOSE Behavior
• The connection is closed whether or not it has an open cursor or pending results.
• ct_close does not behave asynchronously when called with the CS_FORCE_CLOSE option. When ct_close(CS_FORCE_CLOSE) is called to close an asynchronous connection, it returns CS_SUCCEED or CS_FAIL immediately, to indicate the network response. In this case, no completion callback event occurs.
• CS_FORCE_CLOSE is useful when:
- An application is hung, waiting for a server response.
- An application cannot call ct_close(CS_UNUSED) because results are pending.
• Because no logout message is sent to the server, the server cannot tell whether the close is intentional or whether it is the result of a lost connection or crashed client.
• An application can call ct_close(CS_FORCE_CLOSE) when an asynchronous operation is pending.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed. The most common reason for a ct_close failure is pending results on the connection.
CS_PENDING - Asynchronous network I/O is in effect. If asynchronous network I/O is in effect and ct_close is called with option as CS_FORCE_CLOSE, it returns CS_SUCCEED or CS_FAIL immediately to indicate the network response. In this case, no completion callback event occurs.



• An application calls **ct_close** to close a connection and **ct_con_drop** to de-allocate a closed connection. It is an error to de-allocate a connection which has not been closed.

ct_con_drop

Function
De-allocate a CS_CONNECTION structure.

Syntax
CS_RETCODE ct_con_drop(connection)

CS_CONNECTION *connection;

Parameters
connection - A pointer to a CS_CONNECTION structure. A CS_CONNECTION structure contains information about a particular client/ server connection.

Comments
- ct_con_drop de-allocates a CS_CONNECTION structure. All CS_COMMAND structures associated with the CS_CONNECTION are de-allocated.
- A CS_CONNECTION structure contains information about a particular client/server connection.
- Once a CS_CONNECTION has been de-allocated, it cannot be reused. To allocate a new CS_CONNECTION, an application can call ct_con_alloc.
- An application cannot de-allocate a CS_CONNECTION structure until the connection it represents is closed. To close a connection, an application can call ct_close.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed. The most common reason for a ct_con_drop failure is that the connection is still open.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
ct_con_alloc, ct_close, ct_connect, ct_con_props

CS_BUSY - An asynchronous operation is already pending for this connection. Note that ct_close does not return CS_BUSY when called with option as CS_FORCE_CLOSE.

See Also
ct_callback, ct_con_drop, ct_connect, ct_con_props

ct_exit

Function
Exit Client-Library.

Syntax
CS_RETCODE ct_exit(context, option)

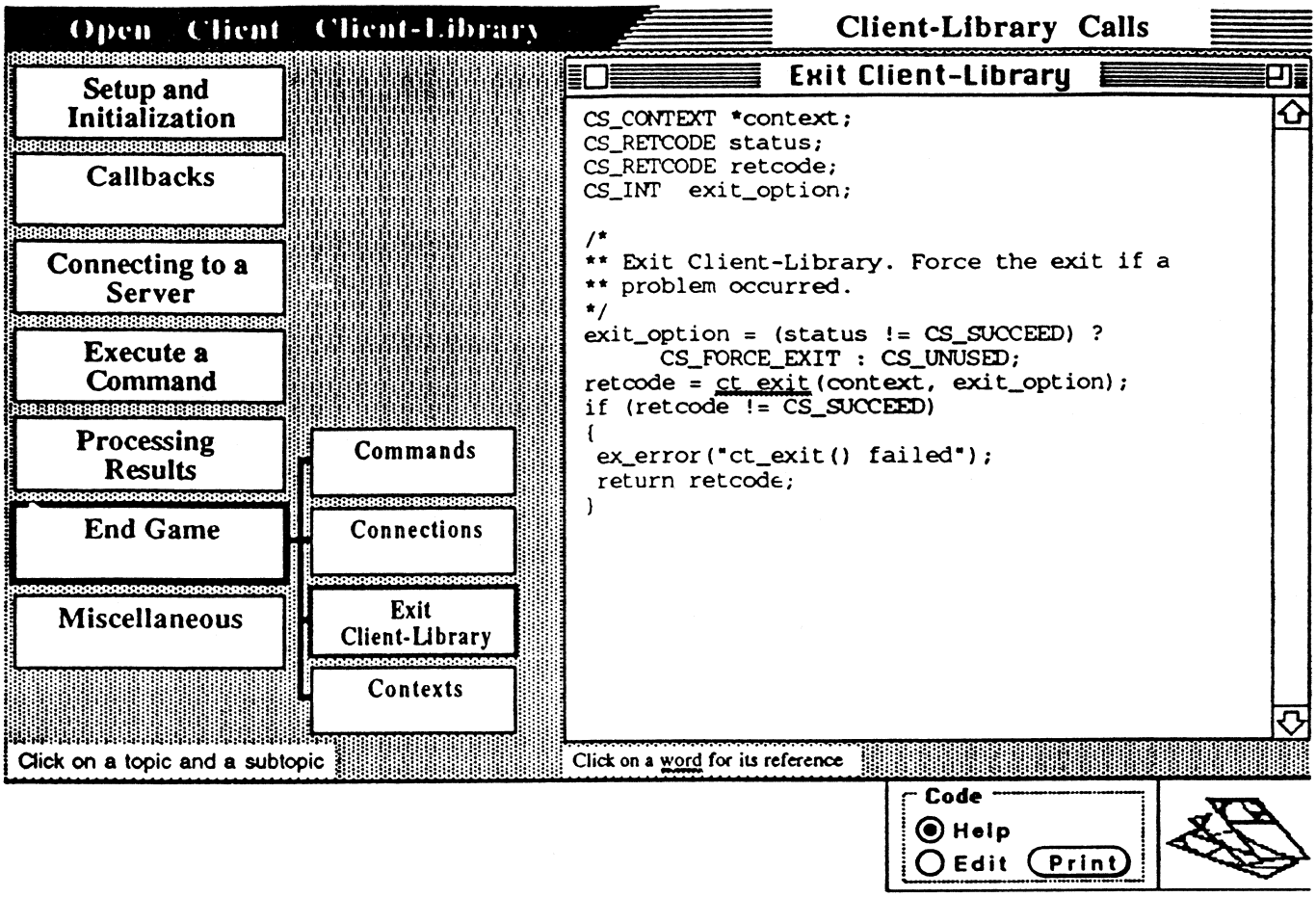
CS_CONTEXT *context;
CS_INT option;

Parameters
context - A pointer to a CS_CONTEXT structure.
context identifies the Client-Library context being exited.
option - ct_exit can behave in different ways, depending on the value specified for option. The following lists the symbolic values that are legal for option:
CS_UNUSED : ct_exit closes all open connections for which no results are pending and terminates Client-Library for this context. If results are pending on one or more connections, ct_exit returns CS_FAIL and does not terminate Client-Library.
CS_FORCE_EXIT : ct_exit closes all open connections for this context, whether or not any results are pending, and terminates Client-Library for this context.

Comments
• ct_exit terminates Client-Library for a specific context. It closes all open connections, de-allocates internal data space and cleans up any platform-specific initialization.
• ct_exit must be the last Client-Library routine called within an Client-Library context.
• If an application finds it needs to call Client-Library routines after it has called ct_exit, it can re-initialize Client-Library by calling ct_init again.
• If results are pending on any of the context's connections and option is not passed as CS_FORCE_EXIT, ct_exit returns CS_FAIL. This means that Client-Library is not correctly terminated and that the application must call ct_exit again after handling the connections' pending results.
• ct_exit always completes synchronously, even if asynchronous network I/O has been specified for any of the context's connections.
• An application can call ct_close to close a single connection.
• If ct_init is called for a context, it is an error to de-allocate the context before calling ct_exit.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.

See Also
ct_close, ct_init



• An application calls **ct_exit** to exit Client-Library for a specific context. **ct_exit** closes and de-allocates any open connections and cleans up internal Client-Library data space. **ct_exit** must be the last Client-Library call for a context.

cs_ctx_drop

Function
De-allocate a CS_CONTEXT structure.

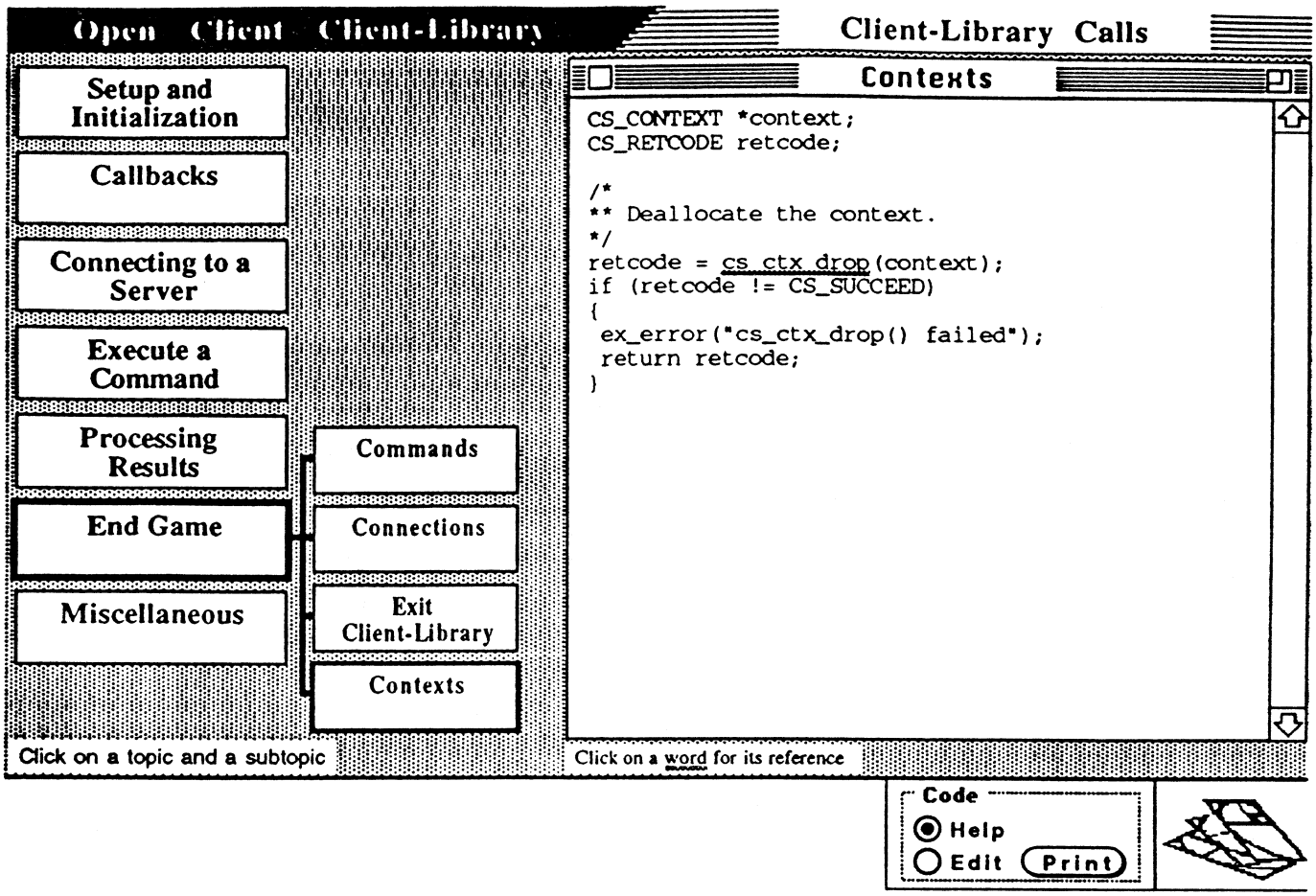
Syntax
CS_RETCODE cs_ctx_drop(context)

CS_CONTEXT *context;

Parameters
context - A pointer to a CS_CONTEXT structure.

Comments
- cs_ctx_drop de-allocates a CS_CONTEXT structure.
- A CS_CONTEXT structure describes a particular context, or operating environment, for a set of server connections.
- Once a CS_CONTEXT has been de-allocated, it cannot be re-used. To allocate a new CS_CONTEXT, an application can call cs_ctx_alloc.
- A Client-Library application cannot call cs_ctx_drop to de-allocate a CS_CONTEXT structure until it has called ct_exit to clean up Client-Library space associated with the context.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed. cs_ctx_drop returns CS_FAIL if the context contains an open connection.
See Also
cs_ctx_alloc, ct_close, ct_exit



- The CS-Library routine **cs_ctx_drop** de-allocates a context structure.

ct_cancel

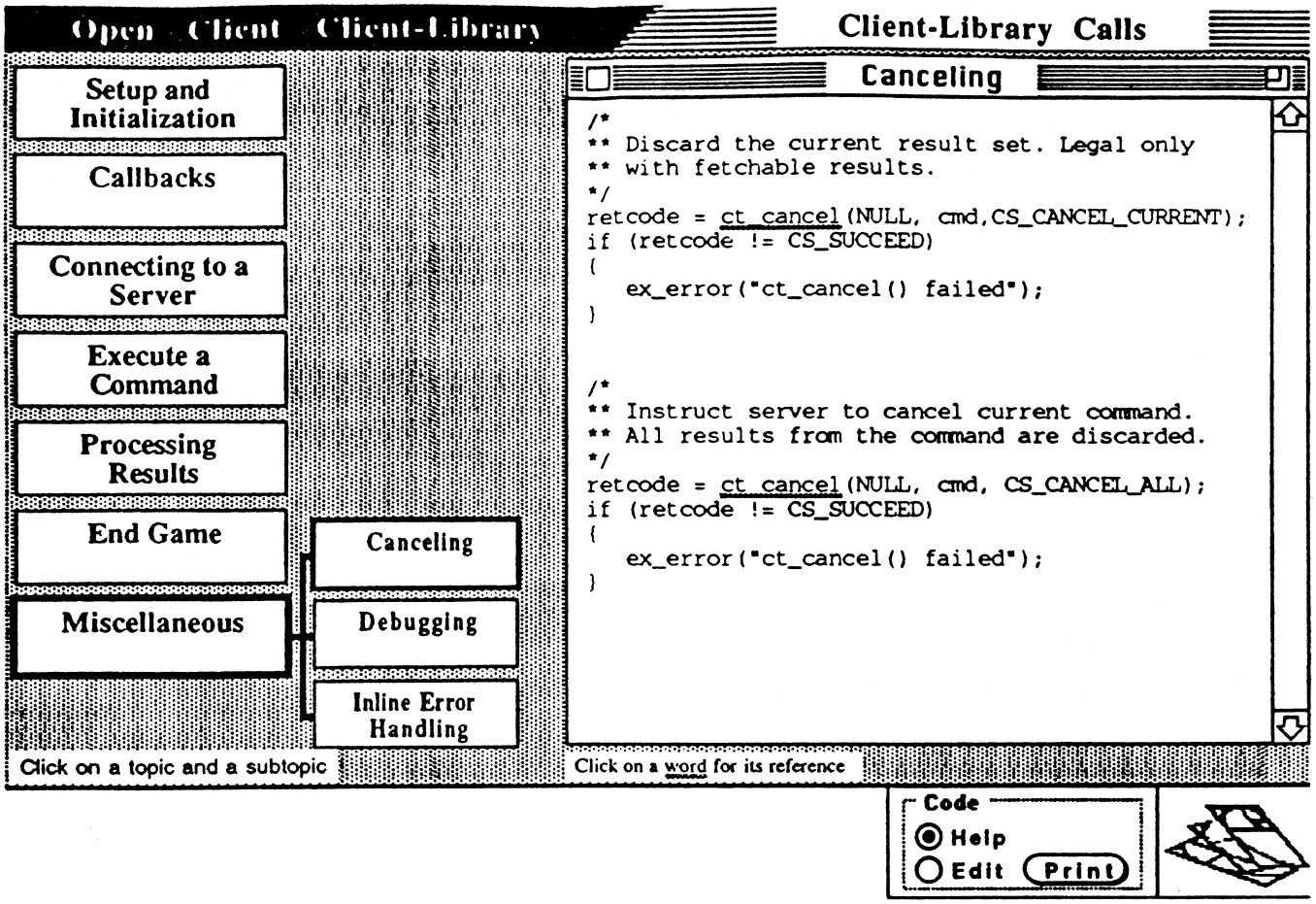
Function
Cancel a command or the results of a command.

Syntax
CS_RETCODE ct_cancel(connection, cmd, type)

CS_CONNECTION *connection;
CS_COMMAND *cmd;
CS_INT type;

Parameters
connection - A pointer to a CS_CONNECTION structure. A CS_CONNECTION structure contains information about a particular client/ server connection. Either connection or cmd must be NULL. If connection is supplied and cmd is NULL, the cancel operation applies to all commands pending for this connection.
cmd - A pointer to the CS_COMMAND structure managing a client/ server operation. Either connection or cmd must be NULL.
For CS_CANCEL_CURRENT cancels, if cmd is supplied and connection is NULL, the cancel operation applies only to the results pending for this command structure. For CS_CANCEL_ATTN and CS_CANCEL_ALL cancels, if cmd is supplied and connection is NULL, the cancel operation applies to all commands pending for this connection.
type - The type of cancel. The following table lists the symbolic values that are legal for type:
CS_CANCEL_ALL : ct_cancel sends an attention to the server, instructing it to cancel the current command. Client-Library immediately discards all results generated by the command. Causes this connection's cursors to enter an undefined state. To determine the state of a cursor, an application can call ct_cmd_props with property as CS_CUR_STATUS.
CS_CANCEL_ATTN : ct_cancel sends an attention to the server, instructing it to cancel the current command. The next time the application reads from the server, Client-Library discards all results generated by the canceled command.
Causes this connection's cursors to enter an undefined state. To determine the state of a cursor, an application can call ct_cmd_props with property as CS_CUR_STATUS.
CS_CANCEL_CURRENT : ct_cancel discards the current result set.
CS_CANCEL_CURRENT is legal only after ct_results has indicated fetchable results. Safe to use on connections with open cursors.

Comments
• ct_cancel cancels the current command or the current result set.
• Canceling a command is equivalent to sending an attention to the server, instructing it to halt execution of the current command. When a command is canceled, any results generated by it are no longer available to an application.
• Canceling a result set is equivalent to discarding the results. Once results are canceled, they are no longer available to an application.
Canceling a Command
• To cancel the current command and all results generated by it, an application calls ct_cancel with type as CS_CANCEL_ATTN or CS_CANCEL_ALL. Both of these calls tell Client-Library to:
- Send an attention to the server, instructing it to halt execution of the current command.
- Discard any results already generated by the command.
• Both types of cancels return CS_SUCCEED immediately if no command is in progress.
• If a command or command batch has been initiated but not yet sent, a CS_CANCEL_ALL cancel discards the initiated command or command batch without sending an attention to the server. If a command or command batch has been initiated but not yet sent, a CS_CANCEL_ATTN cancel has no effect.
• The difference between CS_CANCEL_ALL and CS_CANCEL_ATTN is:
- CS_CANCEL_ALL causes Client-Library to immediately discard the canceled command's results (if any).
- CS_CANCEL_ATTN causes Client-Library to wait until the application attempts to read from the server before discarding the results.
This difference is important because Client-Library must read from the result stream in order to discard results, and it is not always safe to read from the result stream.
It is not safe to read from the result stream from within callbacks or interrupt handlers.
It is safe to read from the result stream anytime an application is running in its main-line code, except when an asynchronous operation is pending.
• CS_CANCEL_ALL leaves the command structure in a "clean" state, available for use in



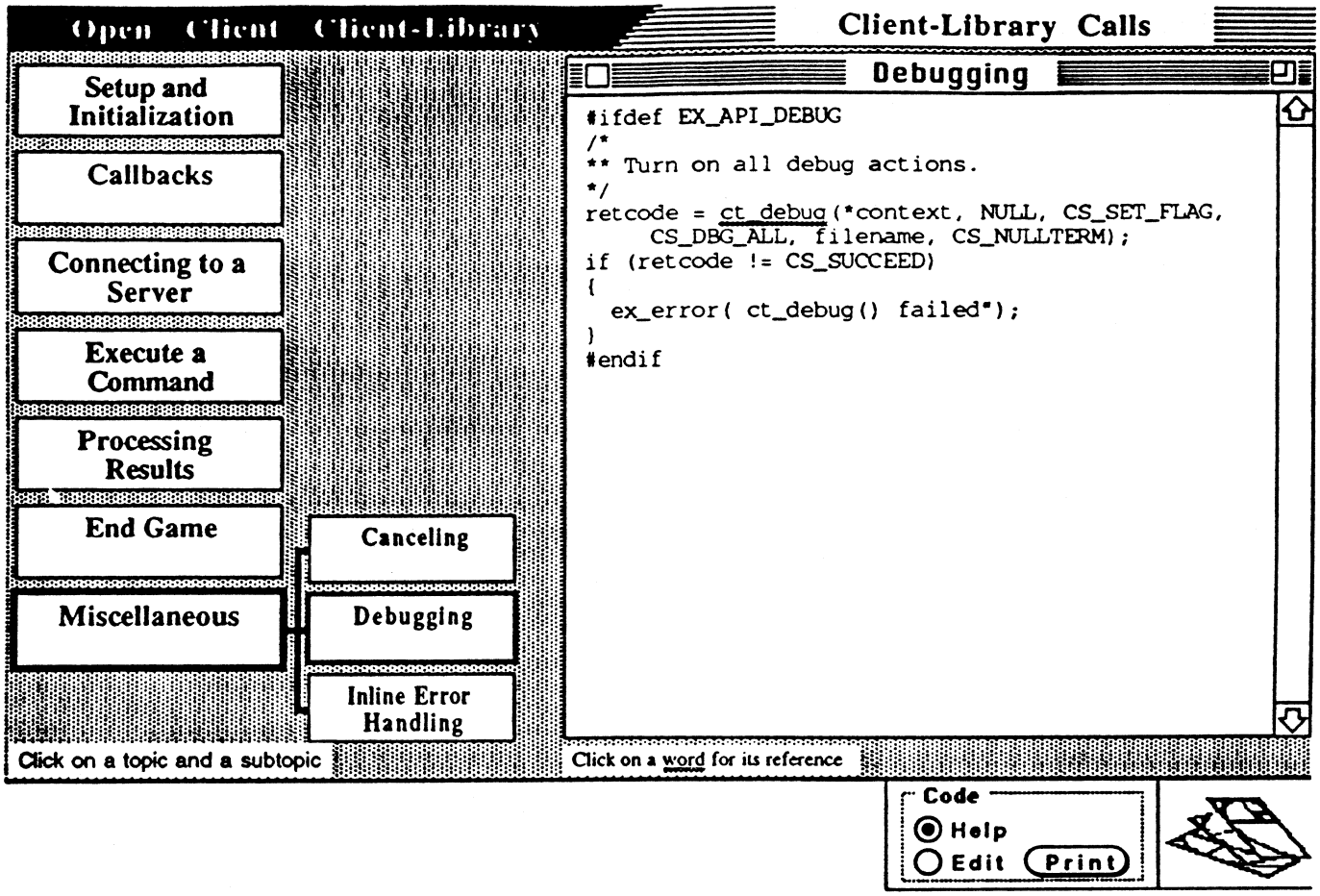
- **ct_cancel** cancels the current command or the current result set.

- Canceling a command is equivalent to sending an attention to the server, instructing it to halt execution of the current command. When a command is canceled, any results generated by it are no longer available to an application.

- Canceling a result set is equivalent to discarding the results. Once results are canceled, they are no longer available to an application.

- To cancel a command set the type for **ct_cancel** to CS_CANCEL_ATTN or CS_CANCEL_ALL.

- To cancel a result set the type for **ct_cancel** to CS_CANCEL_CURRENT.



• **ct_debug** manages debug library operations, allowing an application to enable and disable specific diagnostic subsystems and send the trace information to a file.

• Some debug flags can be enabled only at the connection level, while others can be enabled only at the context level.

• If an application does not specify a file, ct_debug writes debug information to stdout and protocol-form debug information to connect.dat in the application's working directory.

• Some examples of debug actions are:
- CS_DBG_ASYNC : Prints function trace information each time an asynchronous function starts or completes.
- CS_DBG_ERROR : Prints trace information whenever a Client-Library error occurs. This allows a programmer to determine exactly where an error is occurring.
- CS_DBG_MEM : Prints information relating to memory management.

another operation. When a command is canceled with CS_CANCEL_ATTN, however, the command structure cannot be re-used until a Client-Library routine returns CS_CANCELED.
The Client-Library routines that can return CS_CANCELED are:
- ct_cancel(CS_CANCEL_CURRENT)
- ct_cancel(CS_CANCEL_ALL)
- ct_dyndesc
- ct_fetch
- ct_get_data
- ct_options
- ct_recvpassthru
- ct_results
- ct_send
- ct_sendpassthru

• CS_CANCEL_ATTN has two primary uses:
- To cancel commands from within an application's interrupt handlers or callback routines.
- In asynchronous applications, to cancel pending calls to the result-processing routines ct_results and ct_fetch.

• Canceling commands on a connection that has an open cursor may affect the state of the cursor in unexpected ways. For this reason, it is recommended that the CS_CANCEL_ALL and CS_CANCEL_ATTN types of cancels not be used on connections with open cursors. Instead of canceling a cursor command, an application can simply close the cursor.

Canceling the Current Result Set

• To cancel the current result set, an application calls ct_cancel with type as CS_CANCEL_CURRENT. This call tells Client-Library to discard the current result set. It is the equivalent of calling ct_fetch until it returns CS_END_DATA.
• The next result set, if any, remains available to the application, and the current command is not affected.
• Canceling a result set clears the bindings between the result items and program variables.
• An application can use a CS_CANCEL_CURRENT type of cancel only when results are available to be canceled. That is, an application can use a CS_CANCEL_CURRENT cancel only:
- After ct_results indicates a result set of type CS_COMPUTE_RESULT, CS_CURSOR_RESULT, CS_PARAM_RESULT, CS_ROW_RESULT, or CS_STATUS_RESULT; and
- Before ct_fetch returns CS_END_DATA to indicate that all result rows in a result set have been fetched.

• An application cannot use a CS_CANCEL_CURRENT type of cancel after ct_results indicates CS_MSG_RESULT, CS_ROWFMT_RESULT, or CS_COMPUTEFMT_RESULT results, because these types of result sets contain no fetchable data.

Returns

CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.
CS_PENDING - Asynchronous network I/O is in effect.
CS_CANCELED - The cancel operation was canceled. Only the CS_CANCEL_CURRENT and CS_CANCEL_ALL types of cancels can be canceled.
CS_BUSY - An asynchronous operation is already pending for this connection.
CS_TRYING - A cancel operation is already pending for this connection.

See Also
ct_fetch, ct_results

- Memory reference checks: Client-Library verifies that all memory references, both internal and application-specific, are valid.
- User bind space validation: Client-Library checks application memory areas supplied to ct_bind, ensuring that they are valid and that they don't overlap.
- Data structure validation: each time a Client-Library function accesses a data structure, Client-Library first validates the structure.
- Special assertion checking: Client-Library checks that all array references, including strings, are in bounds.

• Because the debug version of Client-Library performs extensive internal checking, application performance will decrease when the debug library is in use. The level of performance decrease depends on the type and number of tracing subsystems that are enabled. To minimize performance decrease, an application programmer can selectively enable tracing subsystems, limiting heavy tracing to problem areas of code.
• Use of the debug library will change the behavior of asynchronous applications that are experiencing timing problems. In this case, the use of external tracing tools (for example, a network protocol analyzer) is recommended.

Returns
CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed.
CS_BUSY - An asynchronous operation is already pending for this connection.

See Also
Error Handling, Client-Library Messages, ct_callback, ct_con_alloc, ct_diag

ct_debug

Function
Manage debug library operations.

Syntax
```
CS_RETCODE ct_debug(context, connection, operation, flag, filename, fnamelen)

CS_CONTEXT *context;
CS_CONNECTION *connection;
CS_INT operation;
CS_INT flag;
CS_CHAR *filename;
CS_INT fnamelen;
```

Parameters
context - A pointer to a CS_CONTEXT structure. A CS_CONTEXT structure defines a Client-Library application context. When operation is CS_SET_DBG_FILE, context must be supplied and connection must be NULL. When setting or clearing flags, use the chart in the flag parameter section to determine whether or not to supply context.
connection - A pointer to a CS_CONNECTION structure. connection must point to a valid CS_CONNECTION structure, but no actual connection to a server is necessary in order to enable debug operations. When operation is CS_SET_PROTOCOL_FILE, connection must be supplied and context must be NULL. When setting or clearing flags, see the chart in the flag parameter section to determine whether or not to supply connection.
operation - The operation to perform. The following lists the possible symbolic values for operation:
CS_SET_FLAG : Enables the subsystems specified by flag.
CS_CLEAR_FLAG : Disables the subsystems specified by flag.
CS_SET_DBG_FILE : Records the name of the file to which it will write character-format debug information.
CS_SET_PROTOCOL_FILE : Records the name of the file to which it will write protocol-form debug information.
flag - A bit mask representing debug subsystems. The following lists the symbolic values that can make up flag:
CS_DBG_ALL : Takes all possible debug actions.
CS_DBG_API_STATES : Prints information relating to Client-Library function-level state
CS_DBG_ASYNC : Prints function trace information each time an asynchronous function starts or completes.
CS_DBG_DIAG : Prints message text whenever a Client-Library or server message is generated.
CS_DBG_ERROR : Prints trace information whenever a Client-Library error occurs. This allows a programmer to determine exactly where an error is occurring.
CS_DBG_MEM : Prints information relating to memory management.
CS_DBG_PROTCOL : Captures information exchanged with a server in protocol-specific (for example, TDS) format. This information is not human readable.
CS_DBG_PROTOCOL_STATES : Prints information relating to Client-Library protocol-level state transitions.
CS_DBG_WARN : Prints messages relating to any "unusual" internal occurrences, such as unreasonable string lengths.
filename - The full path and name of the file to which ct_debug should write the generated debug information.
fnamelen - The length, in bytes, of filename.

Comments
• ct_debug manages debug library operations, allowing an application to enable and disable specific diagnostic subsystems and send the resultant trace information to files.
• ct_debug functionality is available only from within the debug version of Client-Library, libocdbg.a. When called from within the standard Client-Library, liboc.a, it returns CS_FAIL.
• Some debug flags can be enabled only at the connection level, while others can be enabled only at the context level. The chart for the flag parameter indicates the level at which each flag can be enabled.
• If an application does not call ct_debug to specify debug files, ct_debug writes character-format debug information to stdout and protocol-form debug information to connect.dat in the application's working directory.
• When the debug version of Client-Library is linked in with an application, the following behaviors automatically take place:

ct_diag

Function
Manage in-line error handling.

Syntax
CS_RETCODE ct_diag(connection, operation, type, index, buffer)

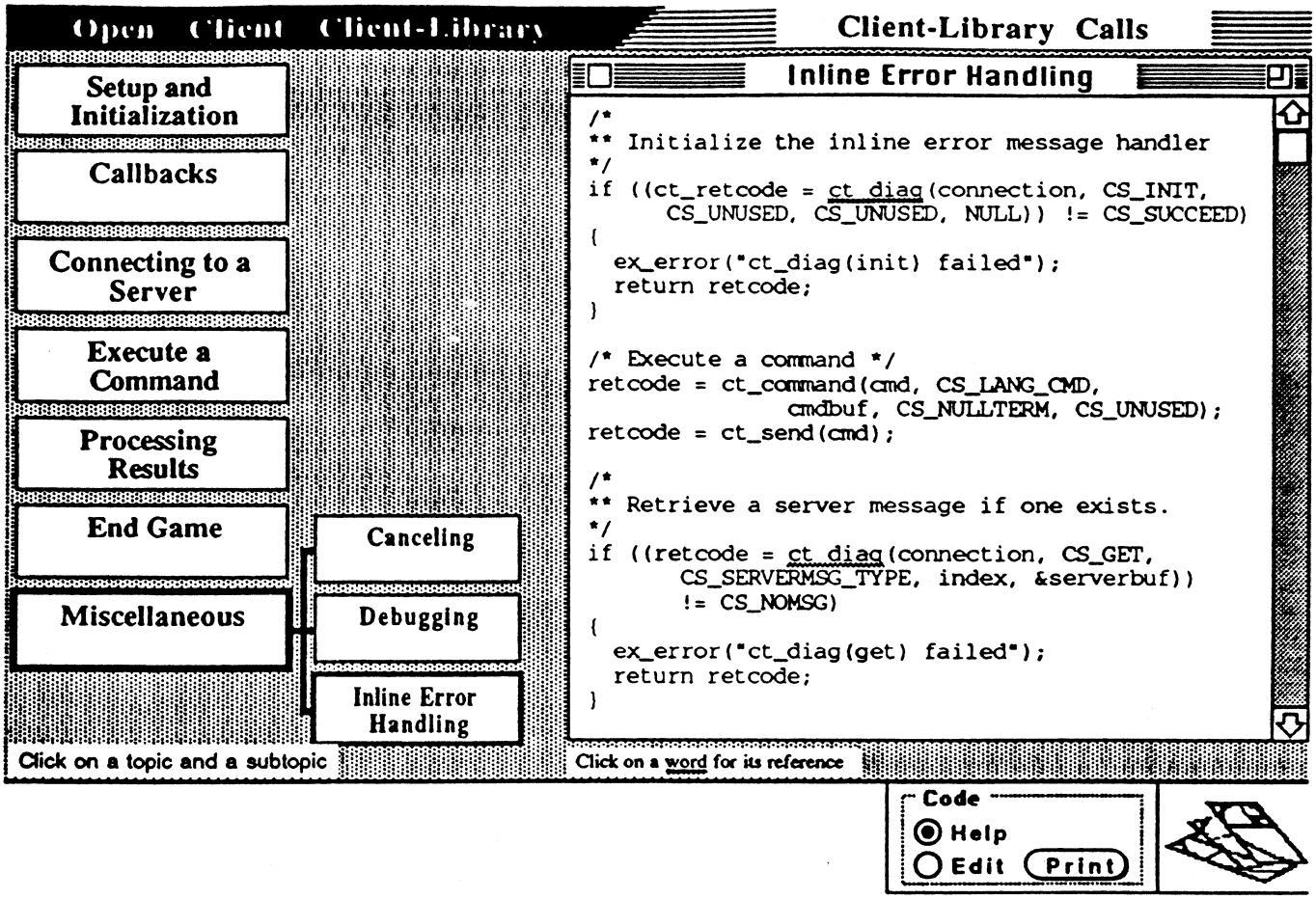
```
CS_CONNECTION *connection;
CS_INT operation;
CS_INT type;
CS_INT index;
CS_VOID *buffer;
```

Parameters
connection - A pointer to a CS_CONNECTION structure. A CS_CONNECTION structure contains information about a particular client/ server connection.
operation - The operation to perform. The following lists the symbolic values that are legal for operation:
CS_INIT : Initializes in-line error handling.
CS_MSGLIMIT : Sets the maximum number of messages to store.
CS_CLEAR : Clears message information for this connection.
CS_GET : Retrieves a specific message.
CS_STATUS : Returns the current number of stored messages.
CS_EED_CMD : Sets *buffer to the address of the CS_COMMAND structure containing extended error data.
CS_SERVERMSG_TYPE : The one-based index of the message for which extended error data is available.
type - Depending on the value of operation, type indicates either the type of structure to receive message information, the type of message on which to operate, or both. The following lists the symbolic values that are legal for type:
SQLCA_TYPE : A SQLCA structure.
SQLCODE_TYPE : A SQLCODE structure.
CS_CLIENTMSG_TYPE : A CS_CLIENTMSG structure. Also used to indicate Client-Library messages.
CS_SERVERMSG_TYPE : A CS_SERVERMSG structure. Also used to indicate server messages.
CS_ALLMSG_TYPE : Client-Library and server messages.
index - The index of the message of interest. The first message has an index of 1, the second an index of 2, and so forth. If type is CS_CLIENTMSG_TYPE, then index refers to Client-Library messages only. If type is CS_SERVERMSG_TYPE, then index refers to server messages only. If type is CS_ALLMSG_TYPE, then index refers to Client-Library and server messages combined.
buffer - A pointer to data space. Depending on the value of operation, buffer can point to a structure or a CS_INT.

Comments
• ct_diag manages in-line message handling.
• A Client-Library application can handle Client-Library and server messages in two ways:
- The application can call ct_callback to install client message and server message callbacks to handle Client-Library and server messages.
- The application can handle Client-Library and server messages in-line, using ct_diag.
It is possible for an application to switch back and forth between the two methods. For information on how to do this, see the Errors and Messages topics page.
• ct_diag manages in-line message handling for a specific connection. If an application has more than one connection, it must make separate ct_diag calls for each connection.
• An application can perform operations on either Client-Library messages, server messages, or both.
For example, an application can clear Client-Library messages without affecting server messages:

```
ct_diag(connection, CS_CLEAR, CS_CLIENTMSG, CS_UNUSED, NULL);
```

• ct_diag allows an application to retrieve message information into standard Client-Library structures (CS_CLIENTMSG and CS_SERVERMSG) or a SQLCA or SQLCODE. When retrieving messages, ct_diag assumes that buffer points to a structure of the type indicated by type. An application that is retrieving messages into a SQLCA or SQLCODE must set the Client-Library property CS_EXTRA_INF to CS_TRUE. This is because the SQL structures require information that is not ordinarily returned by Client-Library's error handling mechanism. An application that is not using the SQL structures can also set CS_EXTRA_INF to CS_TRUE. In this case, the extra information is returned as standard



• ct_diag manages in-line message handling, performing operations on Client-Library messages and/or server messages.

• ct_diag provides the following operations:
- Initializing inline error handling.
- Clearing messages.
- Retrieving messages.
- Limiting Messages.
- Retrieving the number of messages.

CS_SUCCEED - The routine completed successfully.
CS_FAIL - The routine failed. ct_diag returns CS_FAIL if the original error has made the connection unusable.
CS_NOMSG - The application attempted to retrieve a message whose index is higher than the highest valid index.
CS_BUSY - An asynchronous operation is already pending for this connection.

Common reasons for a ct_diag failure include:
- Invalid connection.
- Inability to allocate memory.
- Invalid parameter combination.

See Also
Error and Message Handling, Client-Library Messages, ct_callback, ct_options

Client-Library messages.

Initializing In-Line Error Handling
• To initialize in-line error handling, an application calls ct_diag with operation as CS_INIT.
• Generally, if a connection will use in-line error handling, an application should call ct_diag to initialize in-line error handling for a connection immediately after allocating it.

Clearing Messages
• To clear message information for a connection, an application calls ct_diag with operation as CS_CLEAR.
- To clear Client-Library messages only, an application passes type as CS_CLIENTMSG_TYPE.
- To clear server messages only, an application passes type as CS_SERVERMSG.
- To clear both Client-Library and server messages, pass type as SQLCA, SQLCODE, or CS_ALLMSG_TYPE.
• If type is not CS_ALLMSG_TYPE:
- ct_diag assumes that buffer points to a structure of type type.
- ct_diag clears the *buffer structure by setting it to blanks and/or NULLs, as appropriate.
• Message information is not cleared until an application explicitly calls cs_diag with operations as CS_CLEAR. Retrieving a message does not remove it from the message queue.

Retrieving Messages
• To retrieve message information, an application calls ct_diag with operation as CS_GET, type as the type of structure in which to retrieve the message, index as the one-based index of the message of interest, and *buffer as a structure of the appropriate type.
• If type is CS_CLIENTMSG_TYPE, then index refers only to Client-Library messages. If type is CS_SERVERMSG_TYPE, index refers only to server messages. If type has any other value, index refers to the collective "queue" of both types of messages combined.
• ct_diag fills in the *buffer structure with the message information.
• If an application attempts to retrieve a message whose index is higher than the highest valid index, cs_diag returns CS_NOMSG to indicate that no message is available.
• See the SQLCA, SQLCODE, CS_CLIENTMSG and CS_SERVERMSG topics pages for information on these structures.

Limiting Messages
• Applications running on platforms with limited memory may want to limit the number of messages that Client-Library saves.
• An application can limit the number of saved Client-Library messages, the number of saved server messages, and the total number of saved messages.
• To limit the number of saved messages, an application calls ct_diag with operation as CS_MSGLIMIT and type as CS_CLIENTMSG_TYPE, CS_SERVERMSG_TYPE, or CS_ALLMSG_TYPE:
- If type is CS_CLIENTMSG_TYPE, then the number of Client-Library messages is limited.
- If type is CS_SERVERMSG_TYPE, then the number of server messages is limited.
- If type is CS_ALLMSG_TYPE, then the total number of Client-Library and server messages combined is limited.
• When a specific message limit is reached, Client-Library discards any new messages of that type. When a combined message limit is reached, Client-Library discards any new messages. If Client-Library discards messages, it saves a message to this effect.
• Client-Library's default behavior is to save an unlimited number of messages.

Retrieving the Number of Messages
• To retrieve the number of current messages, an application calls ct_diag with operation as CS_STATUS and type as the type of message of interest.

Getting the CS_COMMAND for Extended Error Data
• To retrieve a pointer to the CS_COMMAND structure containing extended error data (if any), call ct_diag with operation as CS_EED_CMD and type as CS_SERVERMSG_TYPE.
• When an application retrieves a server message into a CS_SERVERMSG structure, Client-Library indicates that extended error data is available for the message by setting the status field in the CS_SERVERMSG structure to CS_EED.
• It is an error to call ct_diag with operation as CS_EED_CMD when extended error data is not available.
• For more information on extended error data, see the Error and Message Handling topics page in Chapter 2 of this manual.

Returns

Appendix

Simple Example Program

```
 drop_connection(con);


 /*
 ** Drop the CS_CONTEXT structure.
 */
 drop_context(context);

}


void
ctx_init(ctxptr)
CS_CONTEXT **ctxptr;
{


 /*
 ** Allocate and initialize a CS_CONTEXT structure.
 */
 if (cs_ctx_alloc(CS_VERSION_100, ctxptr) != CS_SUCCEED)
 {
  error("cs_ctx_alloc failed");
 }

 if (ct_init(*ctxptr, CS_VERSION_100) != CS_SUCCEED)
 {
  error("ct_init failed");
 }


 /*
 ** Install error and message handlers.
 */
 if (ct_callback(*ctxptr, NULL, CS_SET,
  CS_CLIENTMSG_CB, (CS_VOID *)my_err_handler) != CS_SUCCEED)
 {
  error("ct_callback failed");
 }

 if (ct_callback(*ctxptr, NULL, CS_SET,
  CS_SERVERMSG_CB, (CS_VOID *)my_msg_handler) != CS_SUCCEED)
 {
  error("ct_callback failed");
 }

 return;

}


void
open_connection(context, conptr, user, pwd, server)
CS_CONTEXT *context;
CS_CONNECTION **conptr;
CS_CHAR  *user;
CS_CHAR  *pwd;
CS_CHAR  *server;
{

 CS_CONNECTION *local_con;
 CS_INT  *cmd;
```

Example

```
/*
** A Simple Client-Library Application
**
** This example program demontrates the basics for writing
** a synchronous Client-Library application.
**
** Specifically, it demonstrates sending a language command,
** sending a RPC command, sending a parameter to the RPC,
** processing the results, and error handling.
**
*/

#include <stdio.h>
#include <ctpublic.h> /* every CT-Lib application must include this file */


main(argc, argv)
int argc;
char **argv;
{

 CS_CONTEXT *context;
 CS_CONNECTION *con;


 if (argc != 2)
 {
  error("please give server name");
 }


 /*
 ** Allocate and initialize the application's CS_CONTEXT structure.
 */
 ctx_init(&context);


 /*
 ** Open a connection to a server.
 */
 open_connection(context, &con, "sa", NULL, argv[1]);


 /*
 ** Send a query to the server and process the results.
 */
 do_cmd(con);


 /*
 ** Send an rpc to the server and process the results.
 */
 do_rpc(con);


 /*
 ** Close the connection to the server.
 */
```

```
}

 return;
}

void
do_cmd(con)
CS_CONNECTION *con;
{


 CS_COMMAND *cmd;

 /*
 ** Allocate a CS_COMMAND structure.
 */
 if (ct_cmd_alloc(con, &cmd) != CS_SUCCEED)
 {
  error("ct_cmd_alloc failed");
 }


 /*
 ** Send a language command to the server.
 */
 send_lang(cmd);


 /*
 ** Process the results of the command.
 */
 process_results(cmd);


 /*
 ** Deallocate the  CS_COMMAND structure
 */
 if (ct_cmd_drop(cmd) != CS_SUCCEED)
 {
  error("ct_cmd_drop failed");
 }

 return;
}

void
do_rpc(con)
CS_CONNECTION *con;
{

 CS_COMMAND *cmd;

 if (ct_cmd_alloc(con, &cmd) != CS_SUCCEED)
 {
  error("ct_cmd_alloc failed");
 }

 /*
 ** Send RPC command.
 */
 send_rpc(cmd);
```

```
CS_RETCODE connect_ret;
CS_RETCODE ret;
CS_INT  len;
CS_INT  psize;
CS_CHAR  charbuf[32];


/*
** Allocate a CS_CONNECTION structure.
*/
ret = ct_con_alloc(context, conptr);
if (ret != CS_SUCCEED)
{
 error("ct_con_alloc failed");
}
local_con = *conptr;



/*
** Set the user name.
*/
(user == NULL) ? (len = 0) : (len = CS_NULLTERM);
ret = ct_con_props(local_con, CS_SET, CS_USERNAME, user, len, NULL);
if (ret != CS_SUCCEED)
{
 error("ct_con_props(USERNAME) failed");
}



/*
** Set password.
*/
(pwd == NULL) ? (len = 0) : (len = CS_NULLTERM);
ret = ct_con_props(local_con, CS_SET, CS_PASSWORD, pwd, len, NULL);
if (ret != CS_SUCCEED)
{
 error("ct_con_props(PASSWORD) failed");
}


/*
** Set the packet size.
*/
psize = 512;
ret = ct_con_props(local_con, CS_SET, CS_PACKETSIZE,
   &psize, CS_UNUSED, NULL);
if (ret != CS_SUCCEED)
{
 error("ct_con_props(PACKETSIZE) failed");
}


/*
** Connect to the Server.
*/
(server == NULL) ? (len = 0) : (len = CS_NULLTERM);
ret = ct_connect(local_con, server, len);
if (ret != CS_SUCCEED)
{
 error("ct_connect failed");
```

```
 **
 ** Zero out the structure to ensure that the locale field is
 ** initialized to NULL.
 */
 memset(&datafmt, 0, sizeof(datafmt));
 strcpy(datafmt.name, "@loginame");
 datafmt.namelen = strlen(datafmt.name);
 datafmt.datatype = CS_CHAR_TYPE;
 datafmt.status = CS_INPUTVALUE;
 ret = ct_param(cmd, &datafmt, "sa", CS_NULLTERM, (CS_SMALLINT)0);
 if (ret != CS_SUCCEED)
 {
  error("ct_param() failed");
 }

 /*
 ** Send a command to the server.
 */
 if (ct_send(cmd) != CS_SUCCEED)
 {
  error("ct_send failed");
 }

 return;

}


void
process_results(cmd)
CS_COMMAND *cmd;
{


 CS_RETCODE ret;
 CS_INT  res_type;

 /*
 ** Process the results of a command.
 */
 while((ret = ct_results(cmd, &res_type)) == CS_SUCCEED)
 {


  switch ((int)res_type)
  {
    case CS_ROW_RESULT:
   fprintf(stdout, "ROW RESULT\n");
   fetch_results(cmd);
   break;

    case CS_CMD_SUCCEED:
   /*
   ** This means no rows were returned.
   */
   fprintf(stdout, "CMD SUCCEED\n");
   break;

    case CS_CMD_FAIL:
   /*
   ** The server encountered an error while
   ** processing the command.
```

```
 process_results(cmd);

 if (ct_cmd_drop(cmd) != CS_SUCCEED)
 {
  error("ct_cmd_drop failed");
 }

 return;
}


void
send_lang(cmd)
CS_COMMAND *cmd;
{

 /*
 ** Set a command to be sent to the server.
 */
 if (ct_command(cmd, CS_LANG_CMD, "select * from sysdatabases",
    CS_NULLTERM, CS_UNUSED) != CS_SUCCEED)
 {
  error("ct_command failed");
 }


 /*
 ** Send a command to the server.
 */
 if (ct_send(cmd) != CS_SUCCEED)
 {
  error("ct_send failed");
 }


 return;

}

void
send_rpc(cmd)
CS_COMMAND *cmd;
{


 CS_DATAFMT datafmt;
 CS_RETCODE ret;

 /*
 ** Set the rpc to be executed.
 */
 if (ct_command(cmd, CS_RPC_CMD, "sp_who",
    CS_NULLTERM, CS_UNUSED) != CS_SUCCEED)
 {
  error("ct_command failed");
 }



 /*
 ** Set an rpc parameter.
```

```
  case CS_FAIL:
 /*
 ** Something terrible happened.  A call to ct_cancel()
 ** might salvage the connection, but most likely
 ** the connection will need to be closed.
 */
 error("ct_results() return FAIL.");

  default:
 /*
 ** An unexpected value was returned.
 */
 error("ct_result returned unexpected result type");

}


return;

}


void
fetch_results(cmd)
CS_COMMAND *cmd;
{

 CS_RETCODE ret;
 CS_INT  i;
 CS_DATAFMT fmt;
 CS_INT  num_items;
 CS_CHAR  buf[32][128];

 /*
 ** Determine the number of items in the result set.
 */
 ret = ct_res_info(cmd, CS_NUMDATA, &num_items, CS_UNUSED, NULL);
 if (ret != CS_SUCCEED)
 {
  error("ct_res_info(NUMDATA) failed");
 }


 /*
 ** FYI, ct_describe() returns a description of a result set
 ** item.  The description is returned in a CS_DATAFMT structure.
 ** ct_describe() will not be used in this example.
 **
 **  ct_describe(cmd, item_number, datafmt)
 */



 /*
 ** Bind result set items.
 **
 */
 for (i= 1; i <= num_items; i++)
 {
  memset(&fmt, 0, sizeof(fmt));
  fmt.datatype = CS_CHAR_TYPE;
```

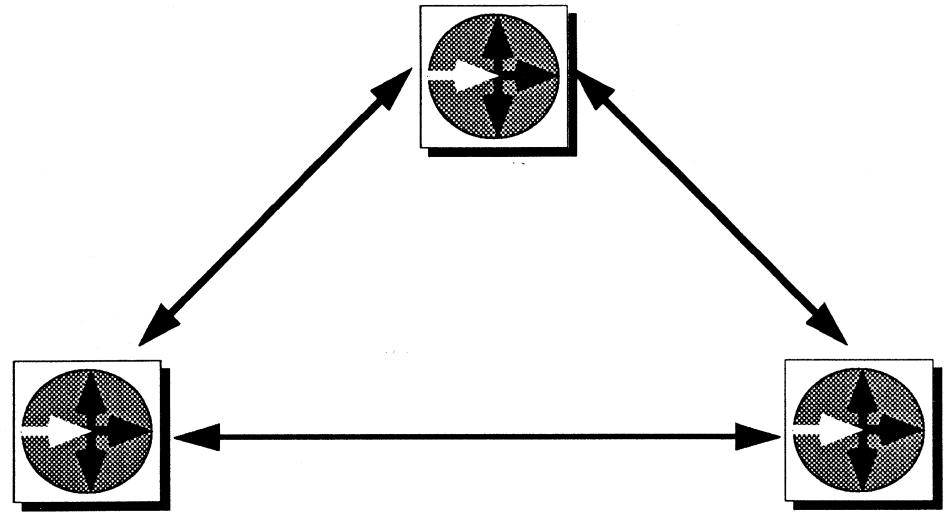
```
 */
 fprintf(stdout, "CMD FAIL\n");
 break;

  case CS_CMD_DONE:
 /*
 ** Done with one result set, let's go
 ** on to the next.
 */
 fprintf(stdout, "CMD DONE\n");
 break;

    case CS_PARAM_RESULT:
 /*
 ** For illustative purposes throw away
 ** the return parameters.
 */
 fprintf(stdout, "PARAM RESULT\n");
 do_cancel_current(cmd);
 break;

    case CS_STATUS_RESULT:
 /*
 ** The same routine can be used to get
 ** status results, as was used to get
 ** row results.
 */
 fetch_results(cmd);
 break;

  default:
 /*
 ** We got something unexpected.  Try to
 ** throw away the result set.
 */
 do_cancel_all(cmd);
 break;
}
}

/*
** Check the return value of ct_results().
**
** We have broken out of the ct_results() loop.  Check
** the return value and act accordingly.
*/
switch((int)ret)
{
  case CS_END_RESULTS:

 /*
 ** Everything went fine.
 */
 fprintf(stdout, "END RESULTS\n");
 break;

  case CS_CANCELED:
 /*
 ** The result set was canceled.
 */
 fprintf(stdout, "CANCELED\n");
 break;
```

```
do_cancel_all(cmd)
CS_COMMAND *cmd;
{

 CS_RETCODE ret;

 ret = ct_cancel(NULL, cmd, CS_CANCEL_ALL);
 if (ret != CS_SUCCEED)
 {
  error("ct_cancel(CURRENT) failed");
 }

 return;
}


void
drop_connection(connection)
CS_CONNECTION *connection;
{

 if (ct_close(connection, CS_UNUSED) != CS_SUCCEED)
 {
  error("ct_close failed");
 }

 if (ct_con_drop(connection) != CS_SUCCEED)
 {
  error("ct_con_drop failed");
 }

 return;

}




void
drop_context(context)
CS_CONTEXT *context;
{

 if (ct_exit(context, CS_FORCE_EXIT) != CS_SUCCEED)
 {
  error("ct_exit failed");
 }

 if (cs_ctx_drop(context) != CS_SUCCEED)
 {
  error("cs_ctx_drop failed");
 }

 return;
}



CS_RETCODE CS_INTERNAL
my_err_handler(context, connection, errmsg)
CS_CONTEXT  *context;
CS_CONNECTION  *connection;
```

```
 fmt.maxlength = 128;
 fmt.count = 1;
 fmt.format = CS_FMT_NULLTERM;

 ret = ct_bind(cmd, i, &fmt, buf[i-1], NULL, NULL);
 if (ret != CS_SUCCEED)
 {
  error("ct_bind() failed");
 }
}

/*
** Read the data.
*/
while (((ret = ct_fetch(cmd, CS_UNUSED, CS_UNUSED,
   CS_UNUSED, (CS_INT *)NULL)) == CS_SUCCEED)
  || (ret == CS_ROW_FAIL))
{
 if (ret == CS_ROW_FAIL)
 {
  fprintf(stderr, "ct_fetch() returned CS_ROW_FAIL\n");
  continue;
 }

 for (i = 1; i <= num_items; i++)
 {
  fprintf(stderr, "%s\t", buf[i-1]);
 }
 fprintf(stderr, "\n");

}


/*
** Check ct_fetch()'s return value.
*/
if (ret != CS_END_DATA)
{
 error("ct_fetch() failed");
}

return;
}


void
do_cancel_current(cmd)
CS_COMMAND *cmd;
{

 CS_RETCODE ret;

 ret = ct_cancel(NULL, cmd, CS_CANCEL_CURRENT);
 if (ret != CS_SUCCEED)
 {
  error("ct_cancel(CURRENT) failed");
 }

 return;
}

void
```

```
CS_CLIENTMSG  *errmsg;
{

 printf("\nOpen Client Message:\n");
 printf("Message number: LAYER = (%ld) ORIGIN = (%ld) ",
  CS_LAYER(errmsg->msgnumber),
  CS_ORIGIN(errmsg->msgnumber));
 printf("SEVERITY = (%ld) NUMBER = (%ld)\n",
  CS_SEVERITY(errmsg->msgnumber),
  CS_NUMBER(errmsg->msgnumber));
 printf("Message String: %s\n", errmsg->msgstring);

 if (errmsg->osstringlen > 0)
 {
  printf("Operating System Error: (%ld): %s\n",
   errmsg->osnumber, errmsg->osstring);
 }

 /*
 ** If there is a serious error, we want to return fail.
 */
 switch ((int)errmsg->severity)
 {
  case CS_SV_INFORM:
  case CS_SV_API_FAIL:
  case CS_SV_RETRY_FAIL:
  case CS_SV_CONFIG_FAIL:
   break;

  case CS_SV_RESOURCE_FAIL:
  case CS_SV_COMM_FAIL:
  case CS_SV_INTERNAL_FAIL:
  case CS_SV_FATAL:
  default:
   return(CS_FAIL);
 }

 return (CS_SUCCEED);
}


/* ARGSUSED */
CS_INT my_msg_handler(context, connection, srvmsg)
CS_CONTEXT *context;
CS_CONNECTION *connection;
CS_SERVERMSG *srvmsg;
{
 fprintf(stderr, "Server message %ld, Severity %ld, State %ld\n",
  srvmsg->msgnumber, srvmsg->severity, srvmsg->state);

 if (srvmsg->svrnlen > 0)
 {
  fprintf(stderr, "Server '%s' ", srvmsg->svrname);
 }

 if (srvmsg->proclen > 0)
 {
  fprintf(stderr, "Procedure '%s' ", srvmsg->proc);
 }

 if (srvmsg->line > 0)
 {
```

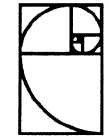
```
  fprintf(stderr, "Line %ld", srvmsg->line);
 }

 fprintf(stderr, "\n\t%s\n", srvmsg->text);

 return(CS_SUCCEED);
}
```



Sybase Replication Server Design Presentation

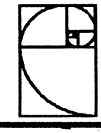
By Sachin Chawla

Client/Server for the online Enterprise



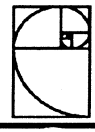
Advantages of Replication

- Improves Performance
  - Offloads Mainframe
  - Reduces Network and Data Server Loads
  - Users and Data Can be Co-located
  - User Transaction Response Time Reduced
  - Scaleable Architecture
- Improves Availability
  - Applications May Continue In Spite of Mainframe, Network and Single/Multiple Site Failures
- Supports the Organization
  - Partitioned Administration and Control
  - Transaction Processing vs Decision Support



Applications Benefiting From Replication

- Geographically Distributed Mission Critical
  - Trading
  - Banking
  - Credit Card
  - Reservations Systems
- Decision Support
  - Local Applications
  - Distributed Applications Supporting Mission Critical

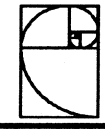


Technical Approaches To Replication

- Tight Consistency
- Loose Consistency

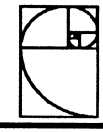
Tight Consistency Replication

- Typically Uses 2 Phase Commit
- All Copies Are Identical
- Replication Is Transparent To Applications
- High Overhead For Protocols
- May Reduce Fault Tolerance, Availability



Two Phase Commit

- Used To Implement Tightly Consistent Distributed Data
- One Transaction Bears the Delay Due to Synchronous Effects of all Servers
- Will Fail if any Component of the System is Unavailable
- Does Not Scale Well - Adding Additional Distributed Components Severely Degrades Performance.
- Appropriate When Absolute Consistency is a Requirement. High Speed Reliable LAN Preferred.

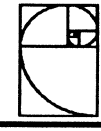


Loose Consistency Replication

- 'Primary' Copy Of Data
- Primary and Replicates Are Not Identical
- Replication Is Not Transparent To Applications
- High Performance - Low Overhead For Protocols
- High Data Availability and Resilience to Failure

Architectures for Loose Replication

- Tape Dump And Reload
- Table Snapshot Replication
- Sybase Online Asynchronous Replication



Tape Dump And Reload

- Database Backups
- Database Logs
- Selective Table Dumps
- Download Data To Remote Sites
- Upload Transactions To Central Site
- Typically Long Latency Period
- Large Volumes of Data Difficult

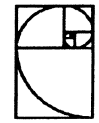
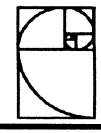


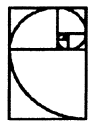
Table Snapshot Replication

- Supported By Some Commercial DBMS
- Table Dump and Reload
- Replication Of File Changes
- May Allow Subsets Of Data
- Proprietary Solutions
- Typically Medium Latency
- Not Transaction Consistent
- No Provision For Remote Update

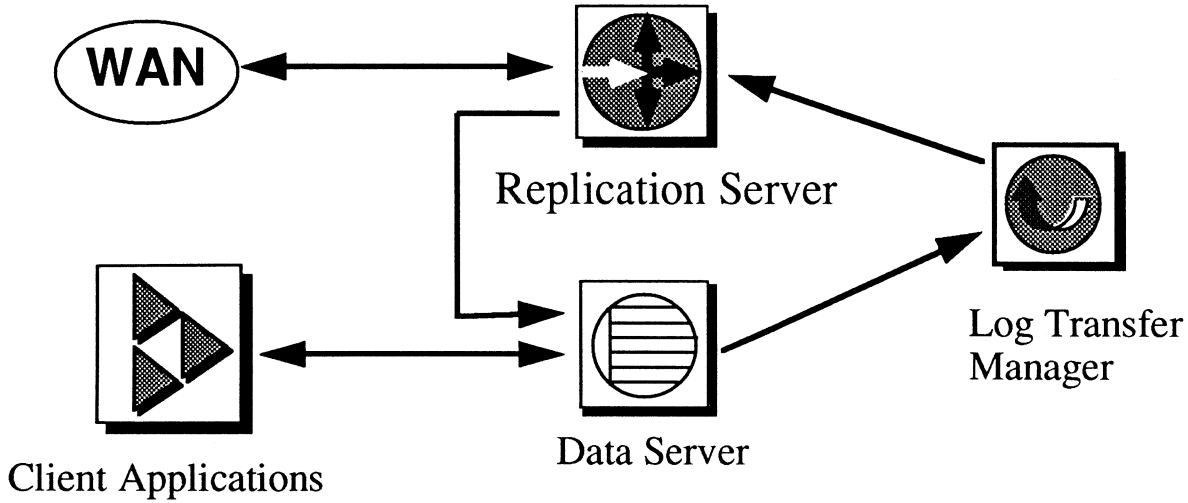


Sybase Online Asynchronous Replication

- Continuous Distribution Of Updates
- Row Level Granularity
- Latency Measured In Seconds
- Maintains Transaction Consistency
- Asynchronous Transaction Processing
- Heterogeneous Open Architecture
- Fault Tolerant Design



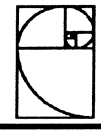
Replicated Data System Components



- Client Application Updates Primary Data
- Data Server Manages Data at Local Site
- Log Transfer Manager Notifies Replication Server of Primary Data Updates
- Replication Server Coordinates Data Replication with Local Data Server and Other Replication Servers

Connecting Components

- Components Use SYBASE Client/Server Interfaces (C/SI)
- Interfaces File at Each Site Defines Local/Remote Replication Servers and Data Servers
- A 'Connection' Defines a Replication Server Connection to a Database
- A 'Route' Defines a Path from One Replication Server to Another

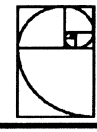


Process for Replicating Data

- Create Database Table at Primary Site, if it does not exist
- Create a Replication Definition to Describe Table
  - Column
  - Data Type
  - Primary Key for Row
  - Columns Used in a Subscription **where** Clause
  - Location of Primary Table
- Create an Empty Table where Data Will be Replicated
- Create Function Strings if Data Server is Not SYBASE SQL Server
- Create Subscriptions at Site where Data is to be Replicated
  - `create subscription affordable for NYSE_Stocks with replicate at Chicago.Stock_db where price >= $20 and Price <= $56`

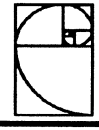
Replicating Stored Procedures

- Create Stored Procedures at Primary Site, if it Does Not Exist
- Create a Replicated User Function Definition
  - Procedure Name
  - Parameters and Data Types
  - Location of Primary Data
- Create Stored Procedures at Replicated Sites
- Create Function Strings if Data Server is not SYBASE SQL Server



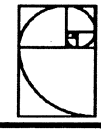
Heterogeneous Data Server Support

- SYBASE SQL Server Fully Supported
- Other Data Servers Required:
  - C/SI Support Through Data Server Directly or Through Open Server Gateway
  - Log Transfer Manager (LTM) Support
  - Error: Class Definitions, Mappings and Processing Actions
  - Function Strings and Function String Classes to Send Directives (i.e. **.insert, delete, update**) to Data Servers

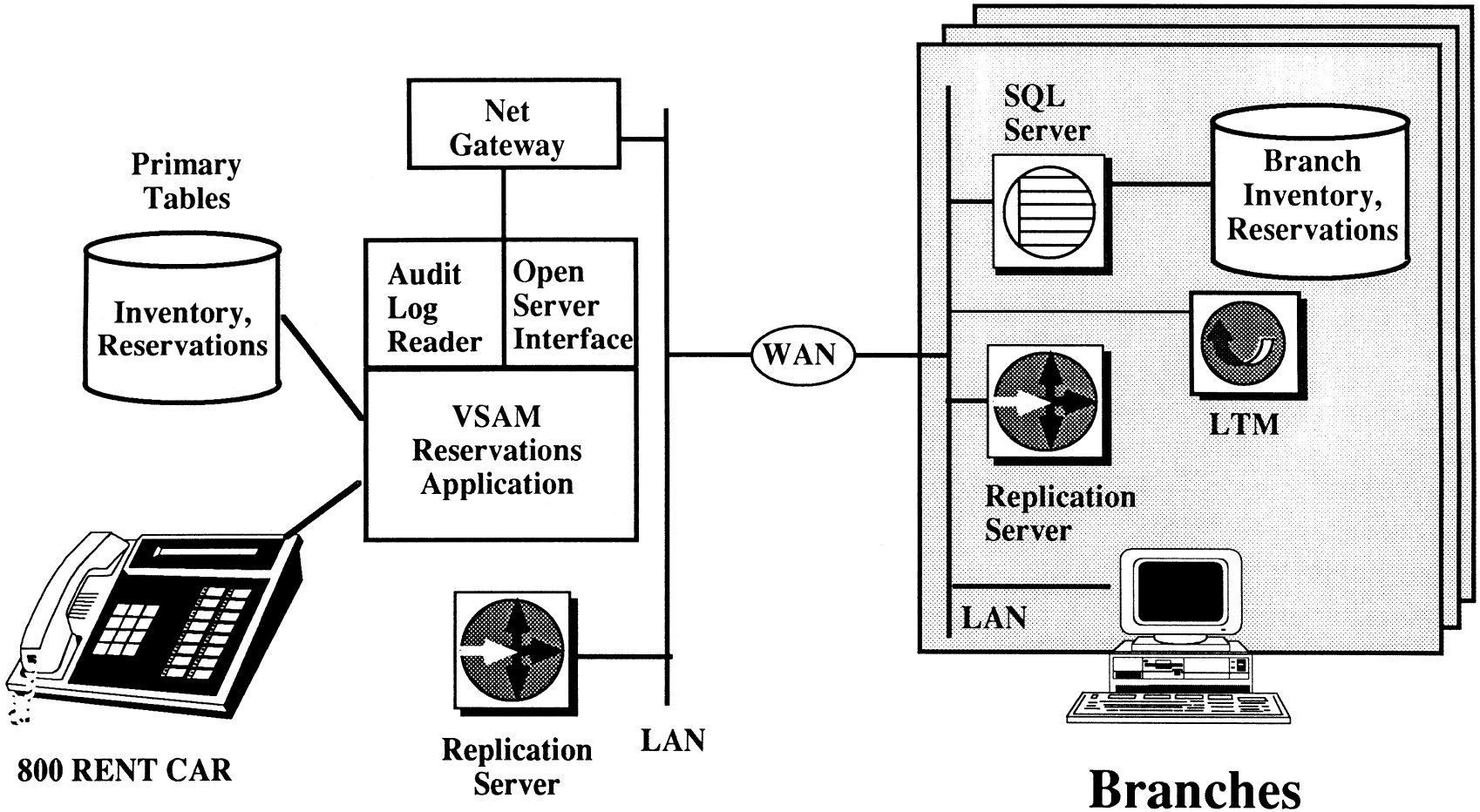


Example Application Scenario

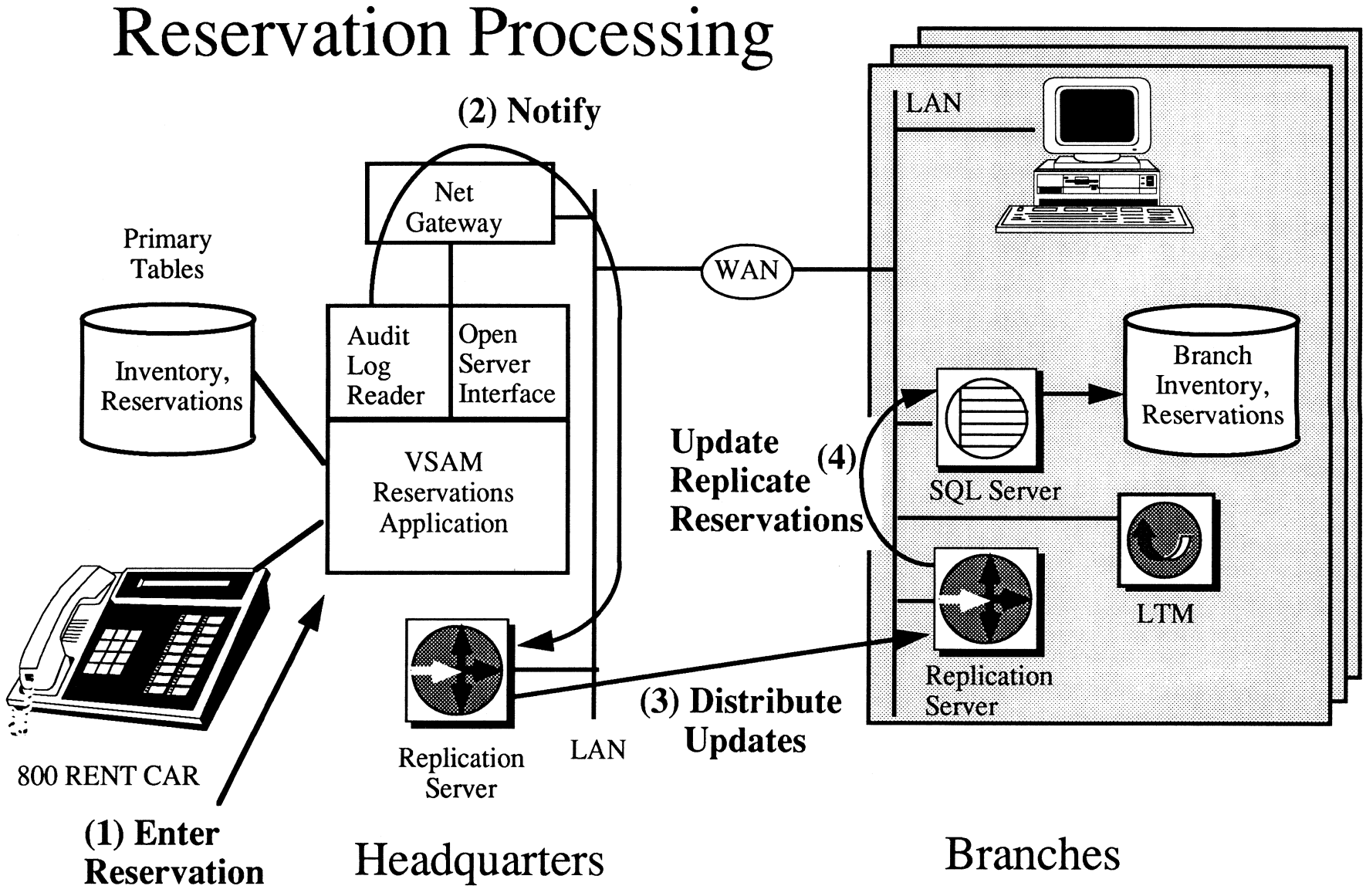
- Application: Rental Car
- Current System
  - Central Mainframe: CICS/VSAM
  - Problems: Old, Overloaded, Expensive, Failure-prone, Keep "Losing" Inventory
- Objectives of New System
  - Increase Performance
  - Increase Reliability
  - Fix Inventory Tracking Problems



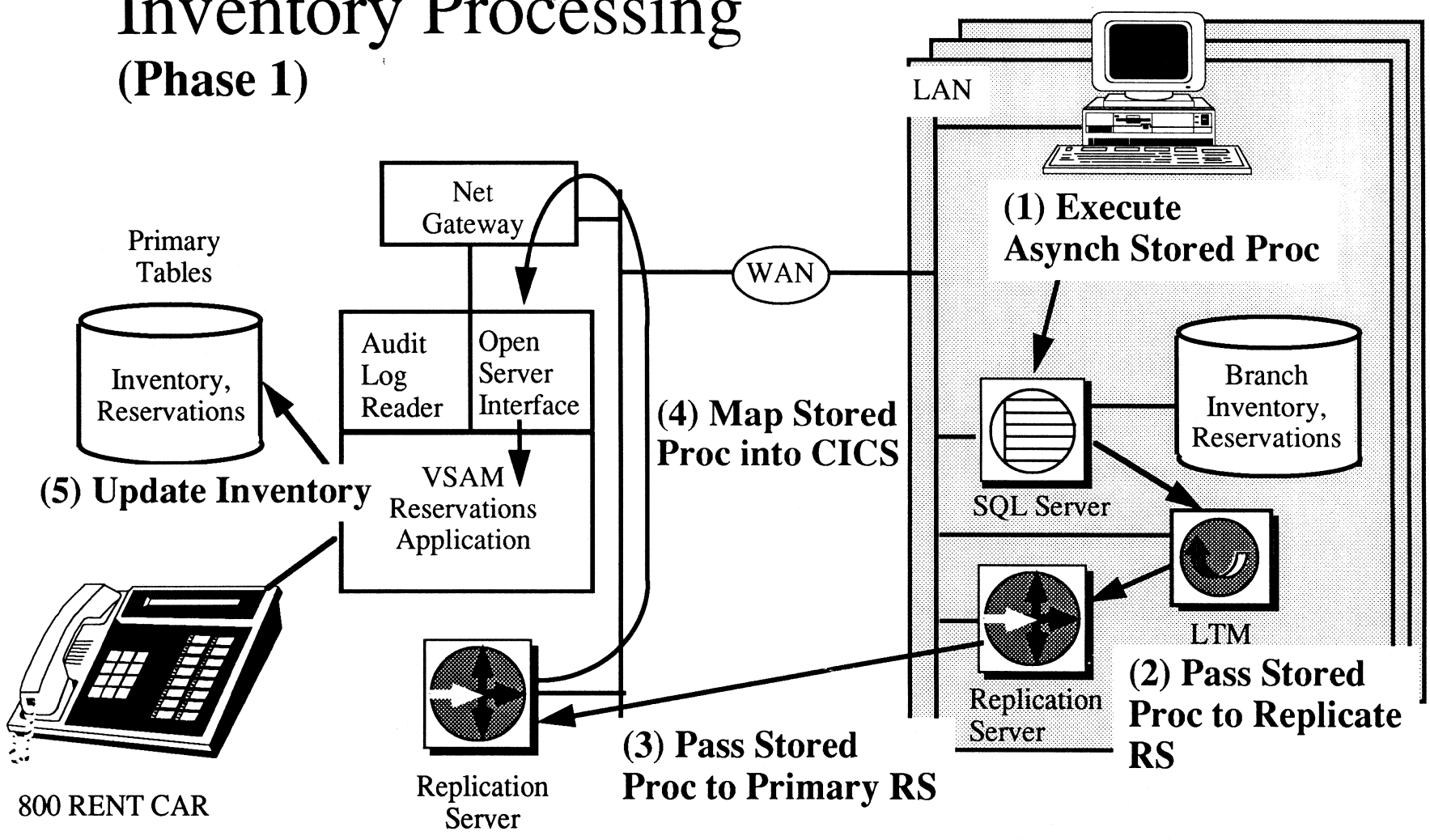
Replication Server Solution



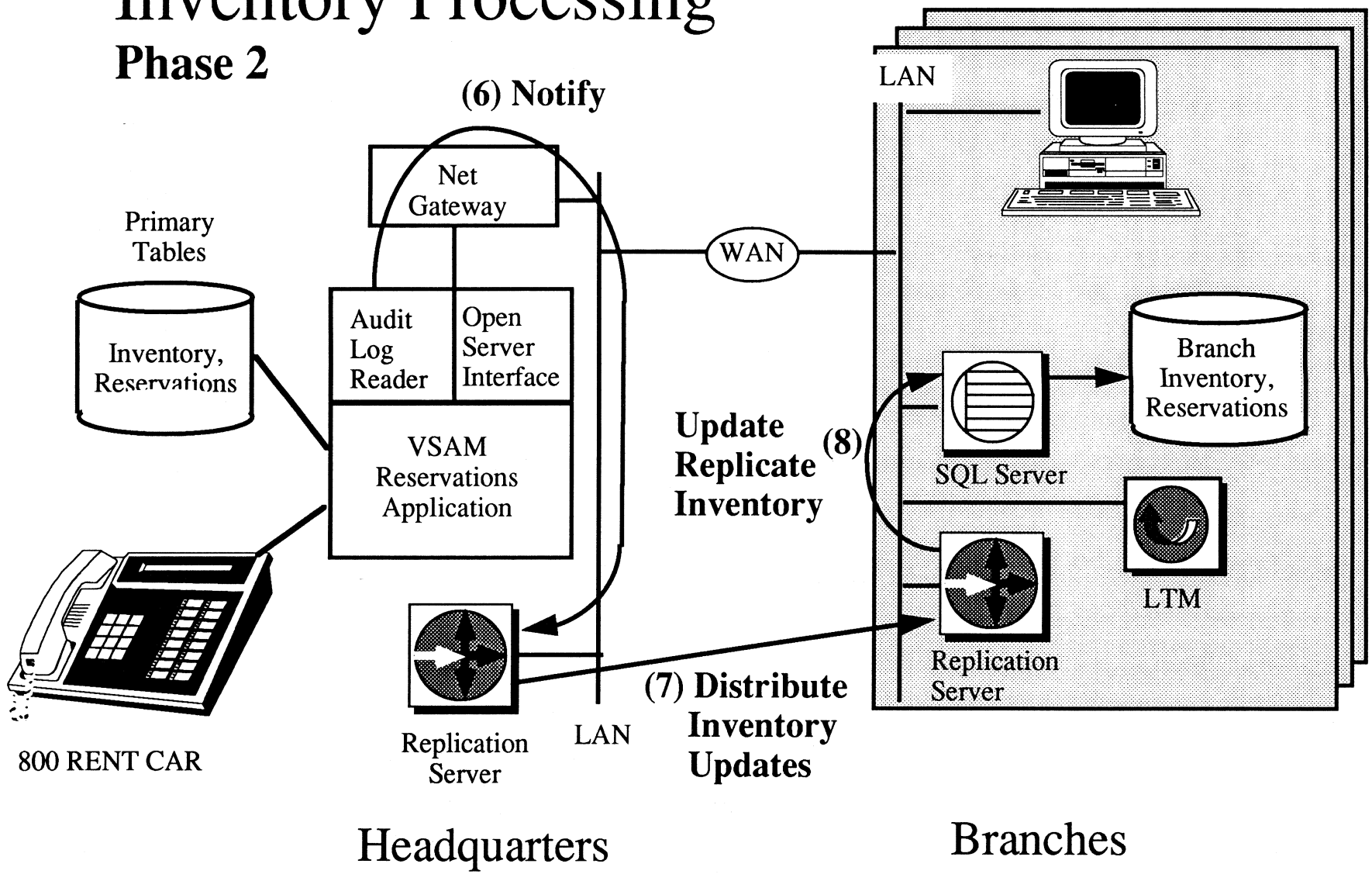
Reservation Processing

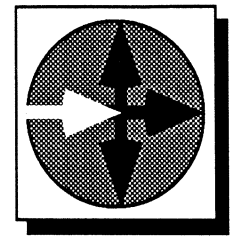


Inventory Processing
(Phase 1)



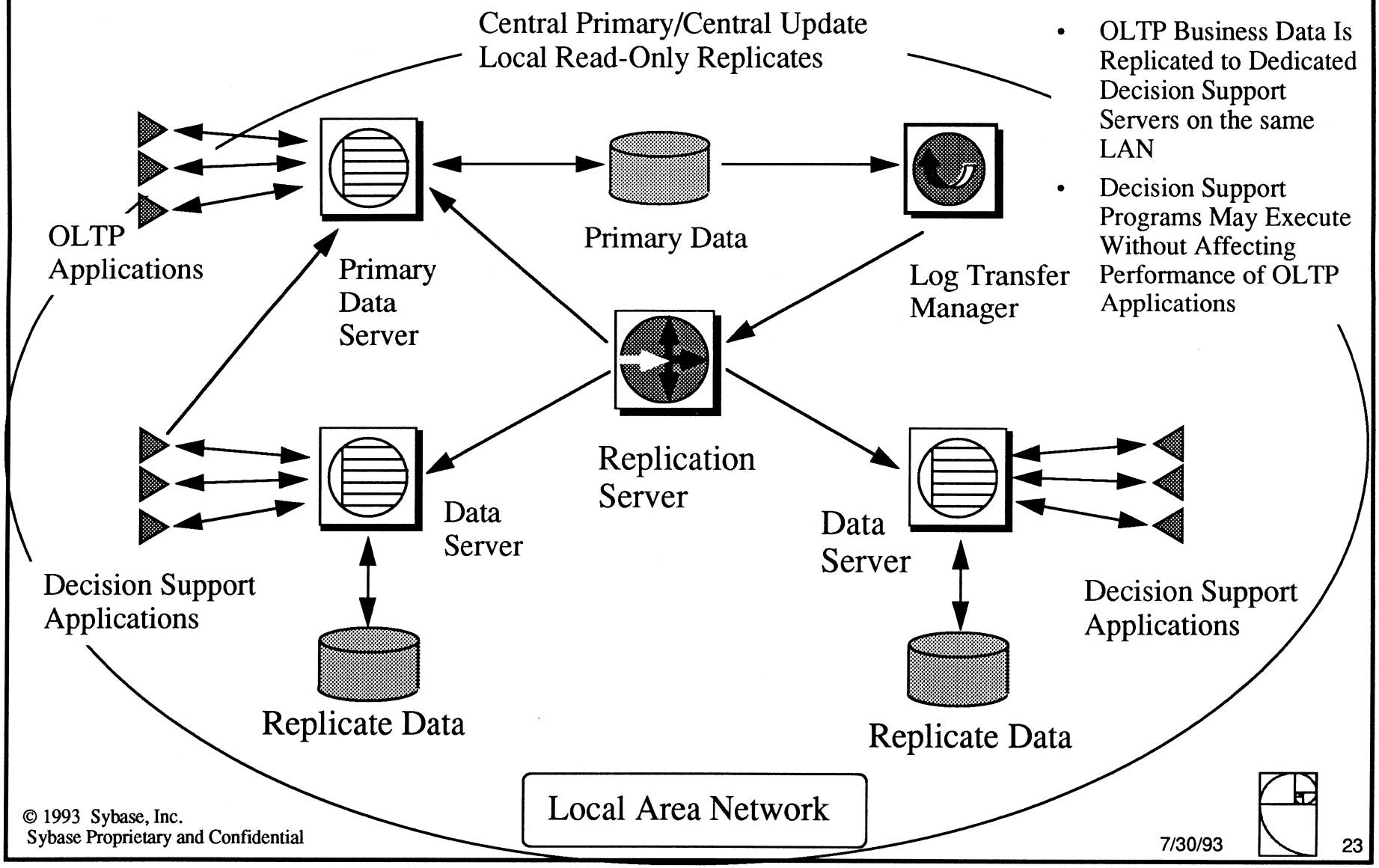
Inventory Processing
Phase 2





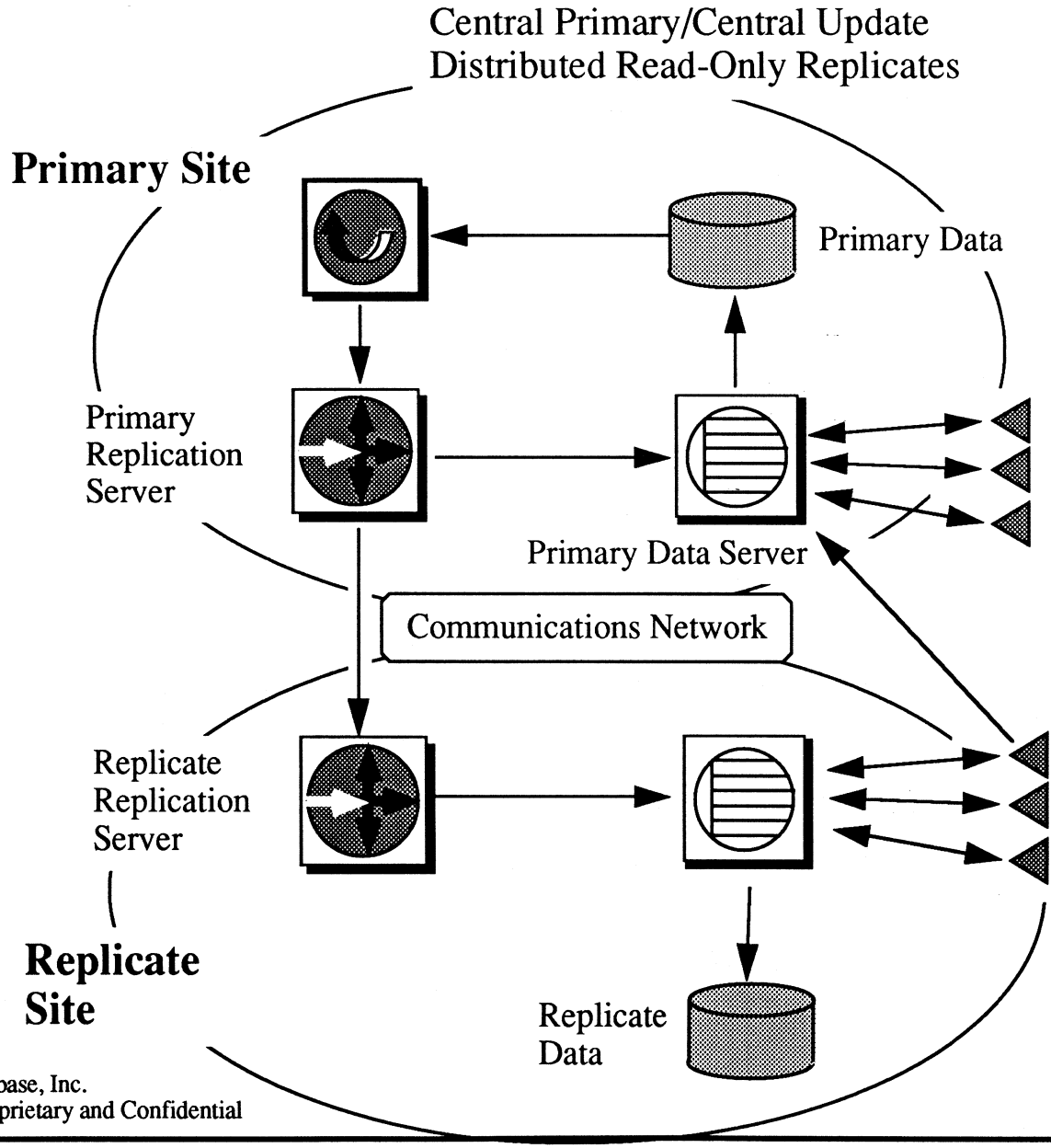
Replication Server Application Architectures

Replication Server Application Architectures



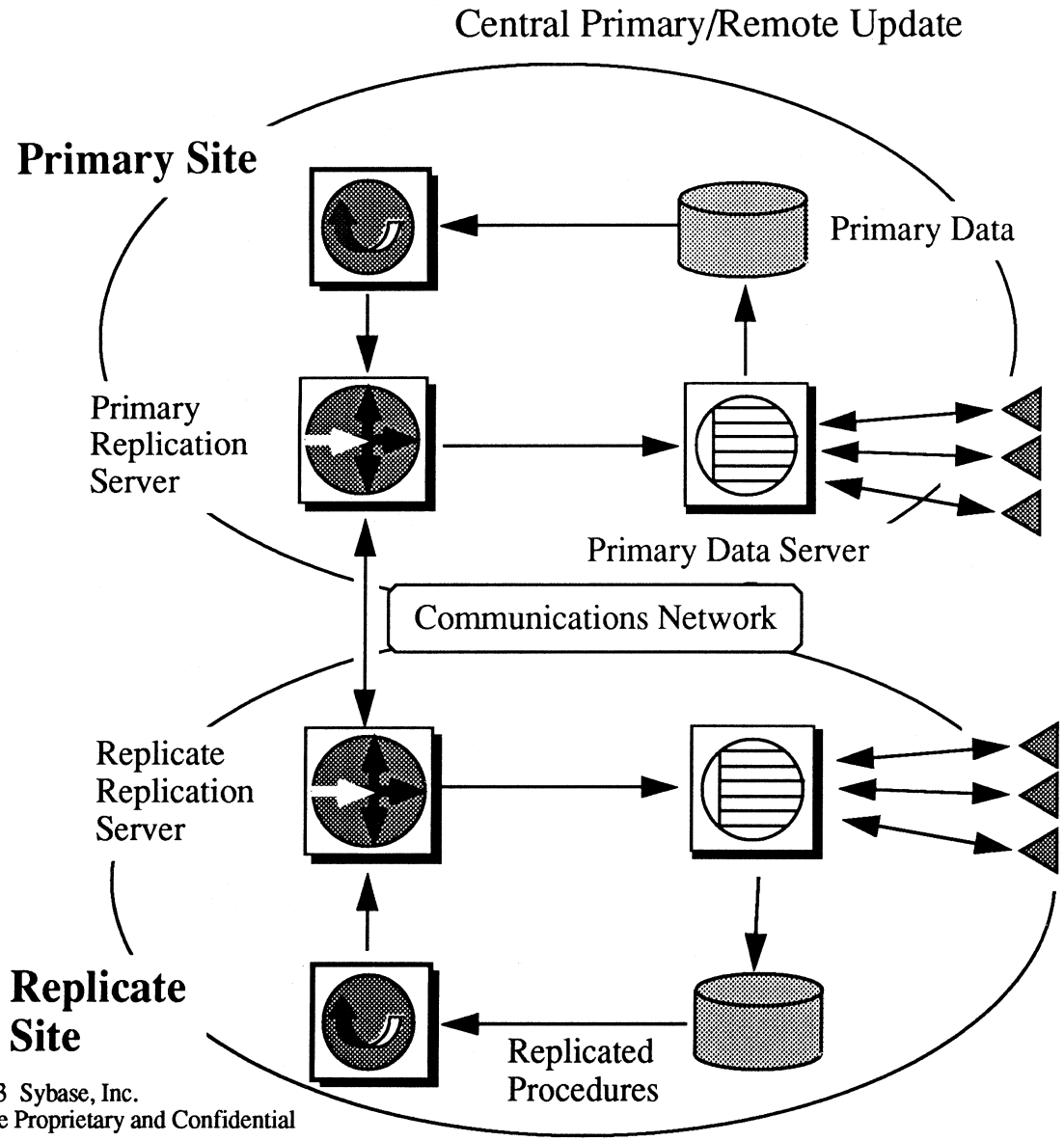
- OLTP Business Data Is Replicated to Dedicated Decision Support Servers on the same LAN
- Decision Support Programs May Execute Without Affecting Performance of OLTP Applications

Replication Server Application Architectures



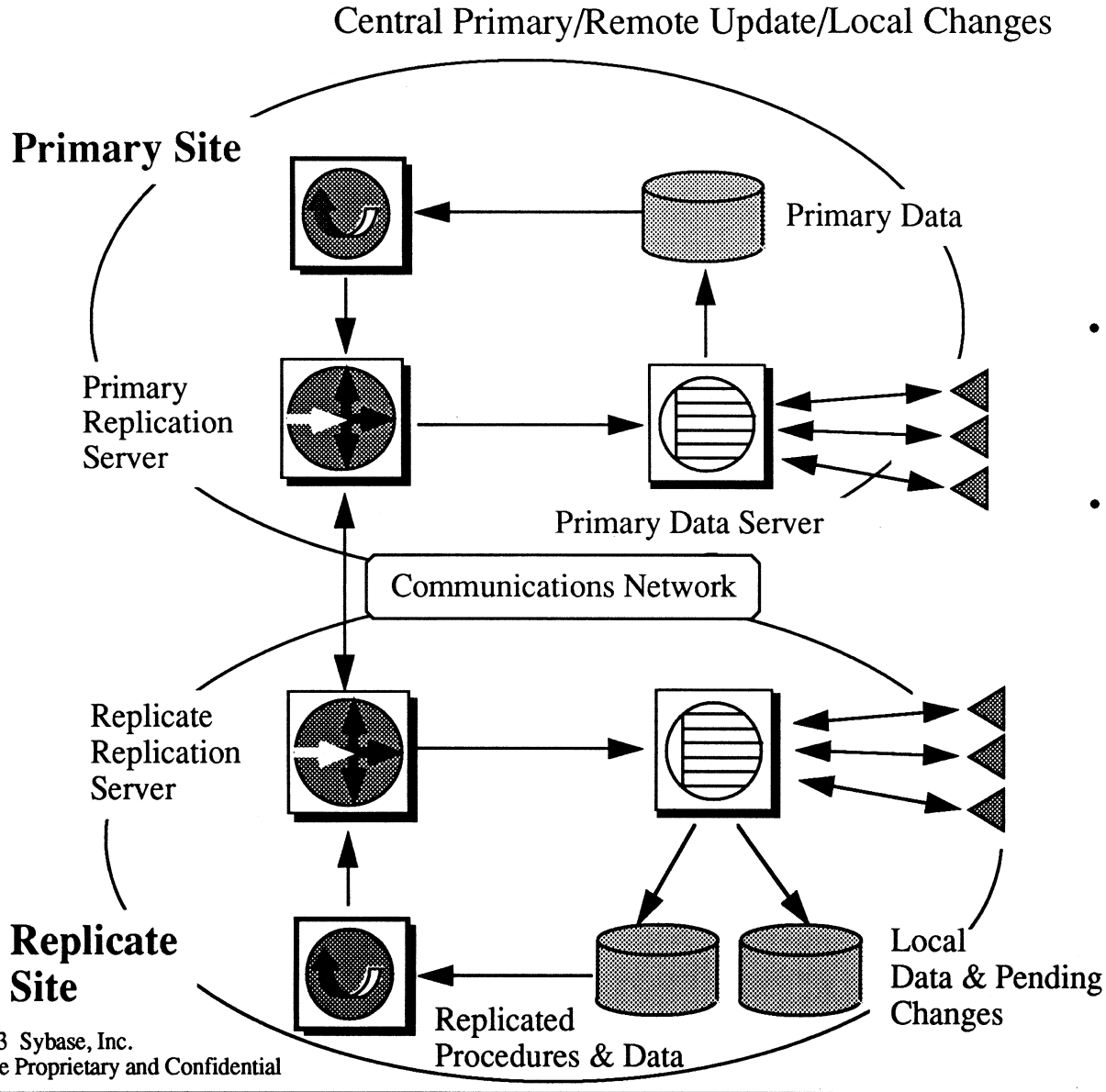
- Remote Applications Use Replicate Data Read-Only, Update Primary Data Directly When Necessary
- Update Requests Must Be Deferred if Primary is not Available

Replication Server Application Architectures



- Remote Applications use Asynchronous Stored Procedures to Update Replicate Data , Changes are not Visible to Remote Applications Until Primary Updates are Distributed.
- Remote Applications may Continue to Submit Updates when Primary is not Available

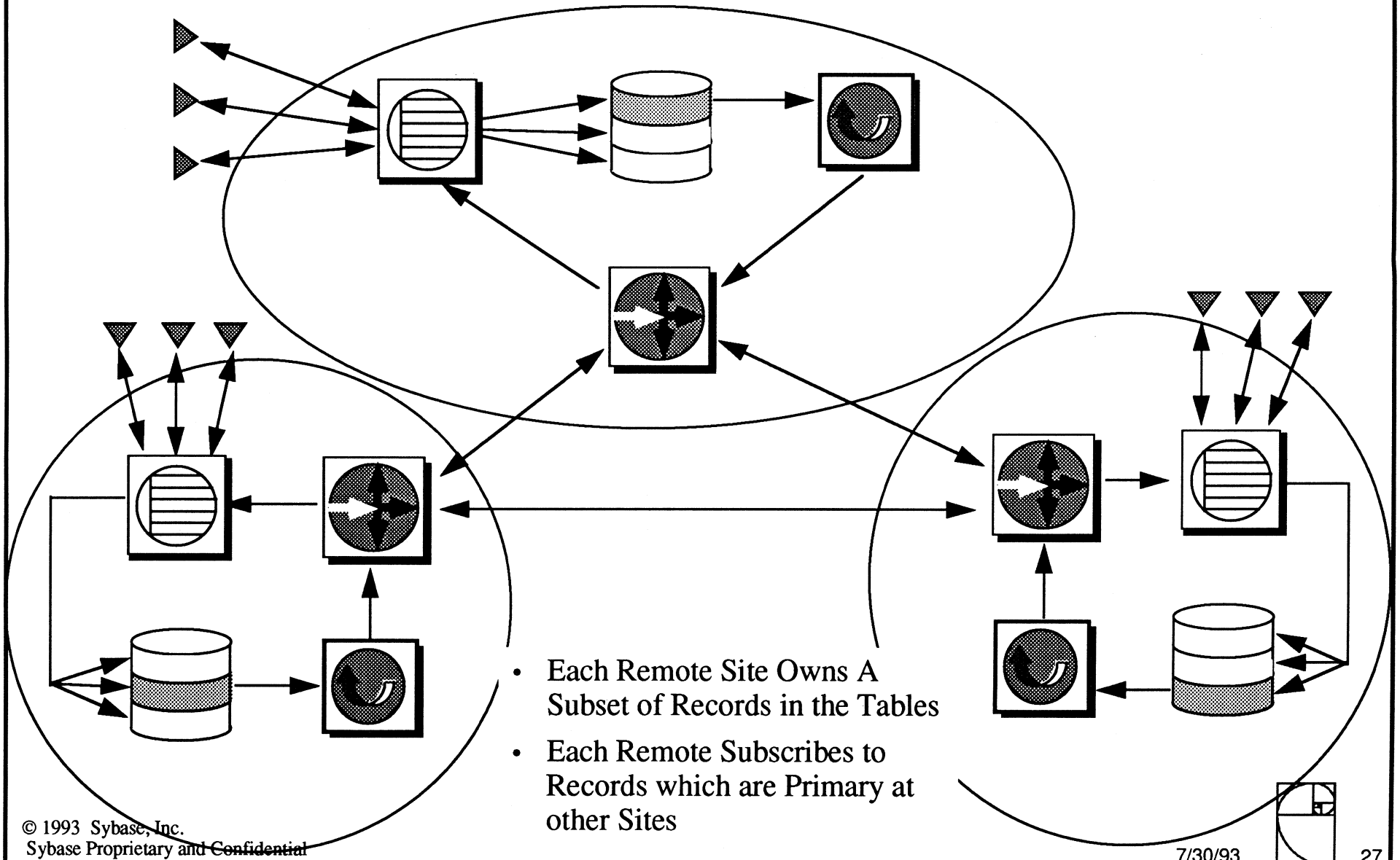
Replication Server Application Architectures



- Remote Applications use Asynchronous Stored Procedures to Update Replicate Data ,
- Pending Changes are Recorded Locally Until Primary Updates are Replicated, Therefore Visible Immediately

Replication Server Application Architectures

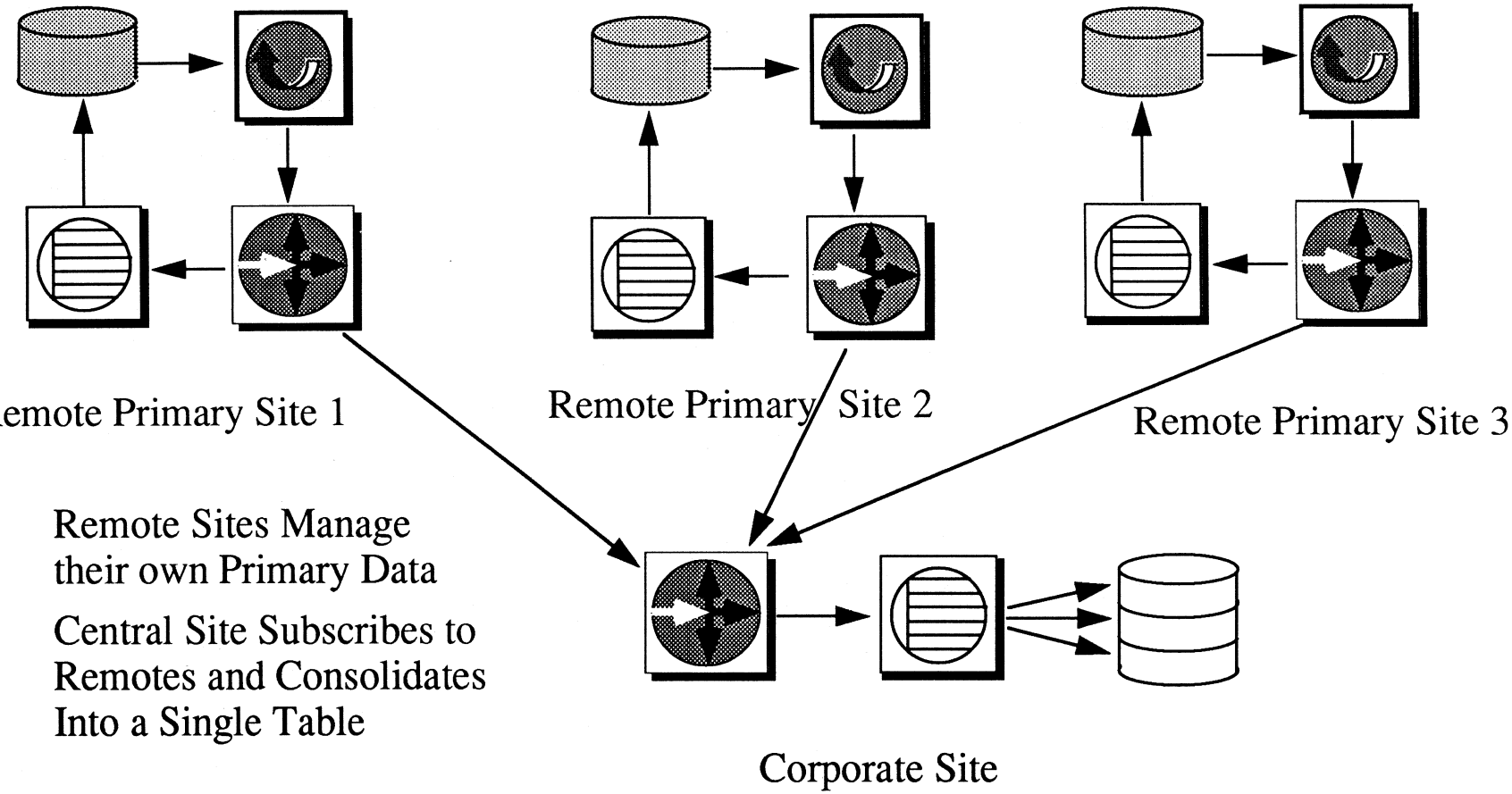
Distributed Partitioned Primary



- Each Remote Site Owns A Subset of Records in the Tables
- Each Remote Subscribes to Records which are Primary at other Sites

Replication Server Application Architectures

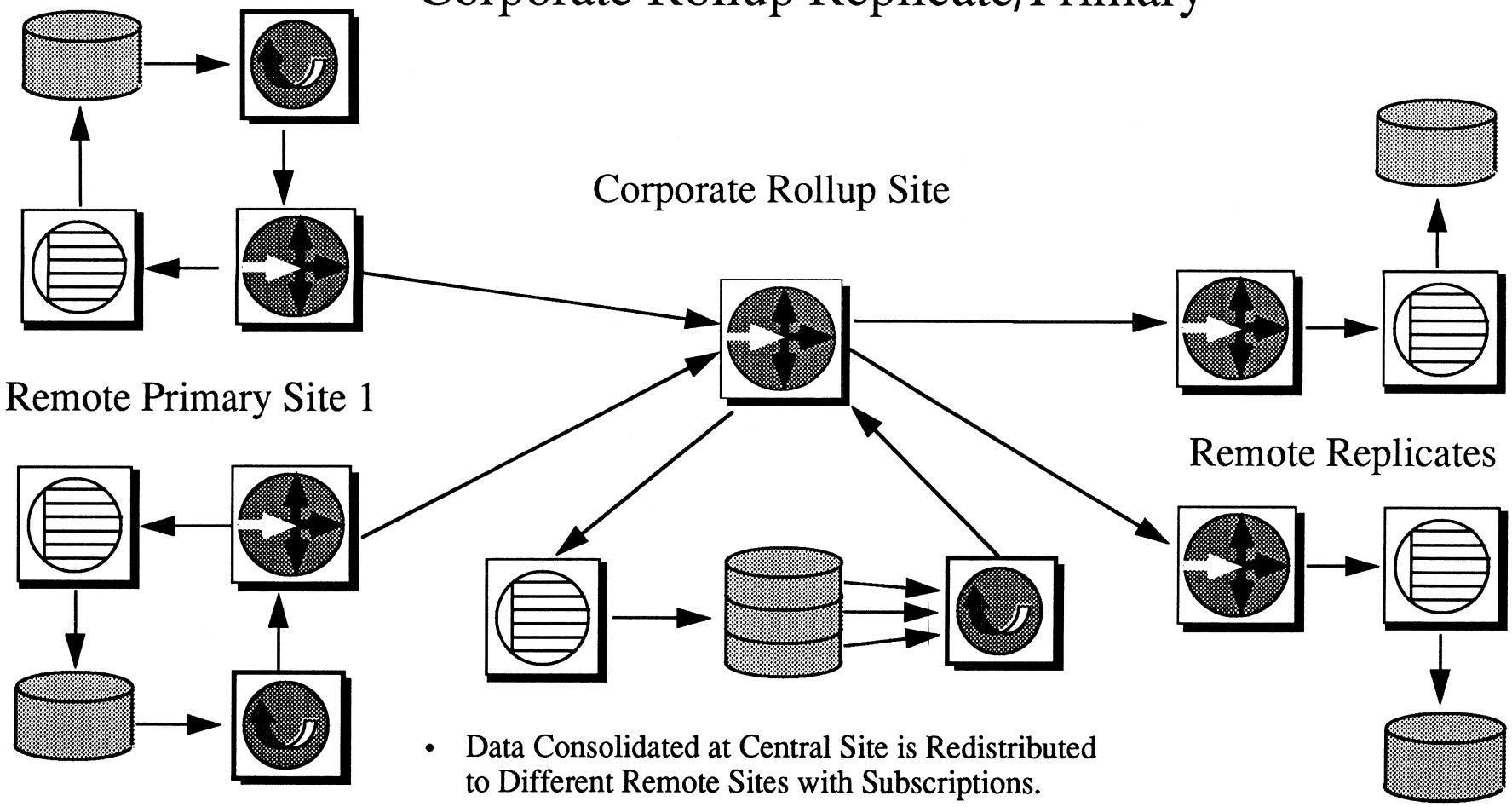
Corporate Rollup Replicate



- Remote Sites Manage their own Primary Data
- Central Site Subscribes to Remotes and Consolidates Into a Single Table

Replication Server Application Architectures

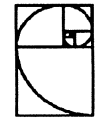
Corporate Rollup Replicate/Primary



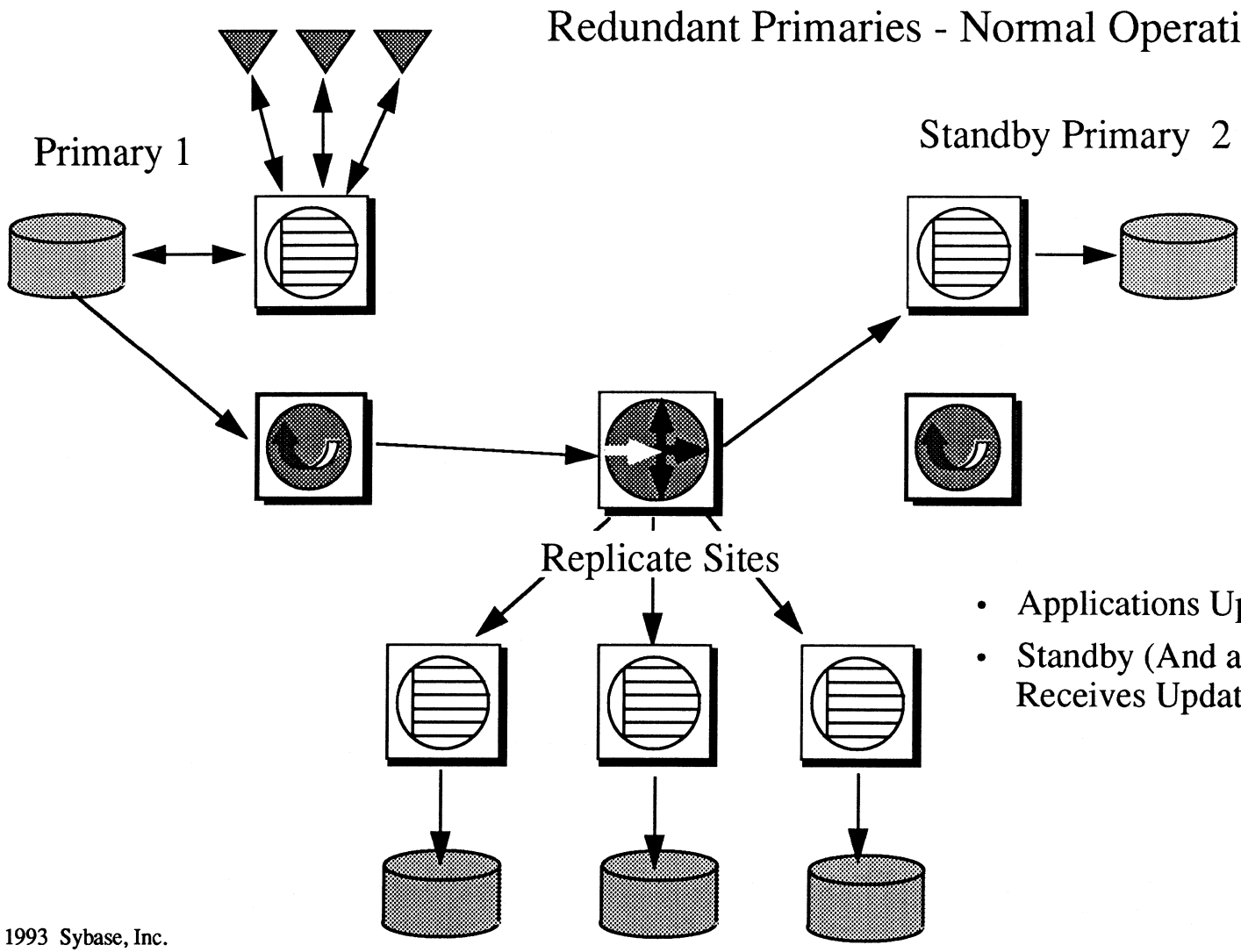
- Data Consolidated at Central Site is Redistributed to Different Remote Sites with Subscriptions.
- Requires two Replication Servers at the Central Site: one which manages the Rollup Table as a Replicate, the Other which Manages the Rollup Table as a Primary

Replication Server Application Architectures
Redundant Primaries

- Used As a Method for Warm Standby
- Also Used To Change Location of Primary At Designated Times
- Tables in Both Primary Databases Are Replicated
- Tables in Both Primary Databases Subscribe to Each Other
- Replicate Sites Must Subscribe to Both Primary Tables
- Applications Must use only one Database as the Primary at a Time
- Applications Must Reconcile Lost Transactions In Case of Primary Switchover



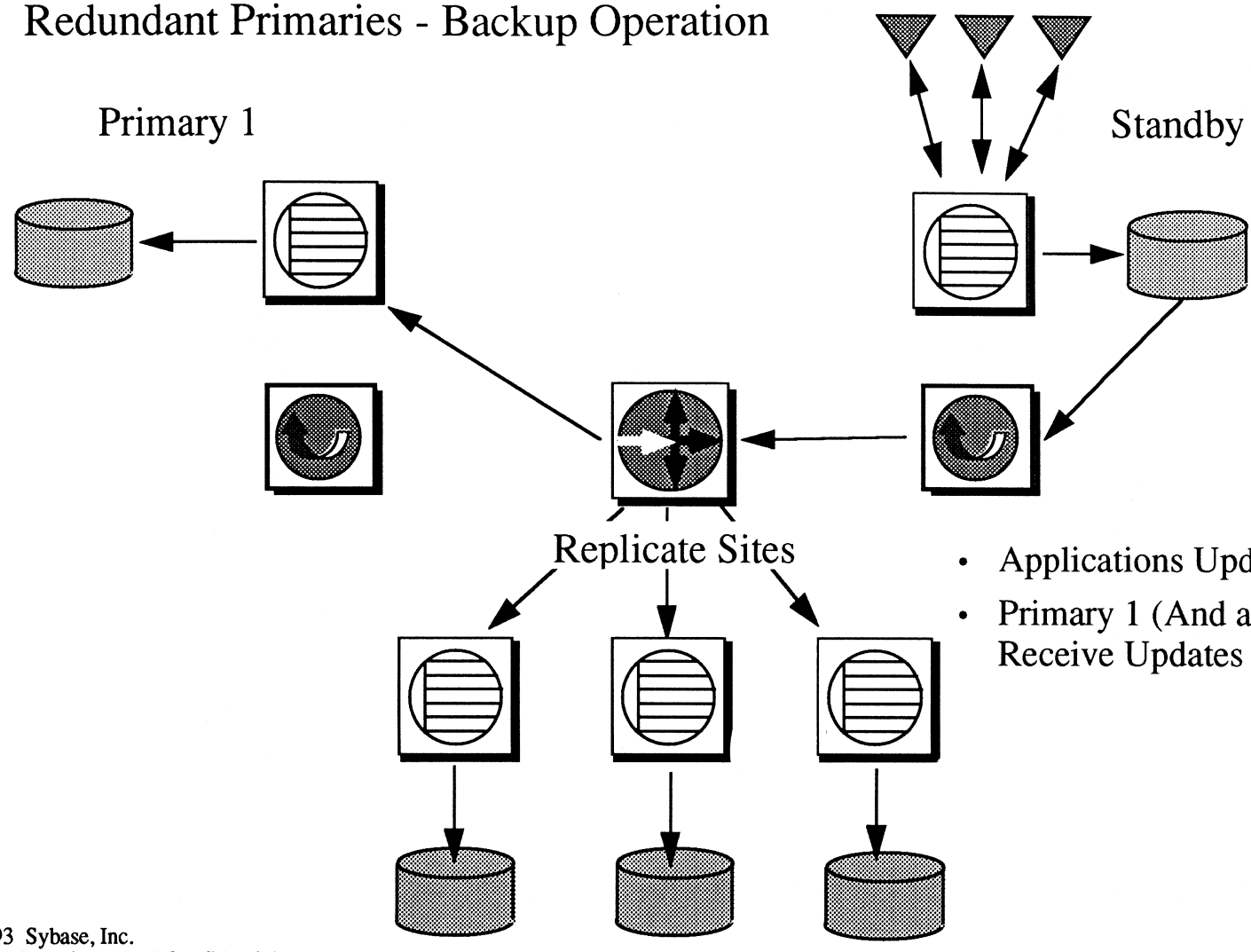
Replication Server Application Architectures



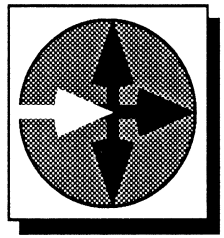
- Applications Update Primary 1
- Standby (And all other Replicates) Receives Updates from Primary 1

Replication Server Application Architectures

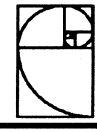
Redundant Primaries - Backup Operation



- Applications Update Primary 2
- Primary 1 (And all Replicates) Receive Updates from Primary 2

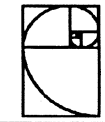


Replication Server: Design Considerations



Performance Factors

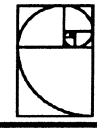
- Number of Primary Databases
- Primary Database Update Rate
- Number of Replicate Databases
- Number of Subscriptions
- Number of Columns
- Size of Columns
- Network Bandwidth
- Platform Capacity



Available Platforms

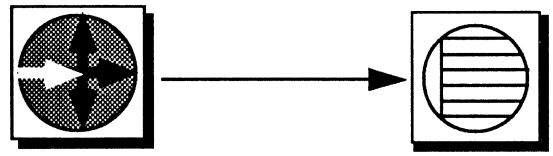
- SQL Server
  - 4.9.1 Platforms
  - System 10 Platforms
- Replication Server and LTM
  - Sun/SunOS
  - RS/6000

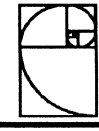
Note: LTM Need Not Run On Same Platform As SQL Server



Replication Server To Replicate Data Server Connections

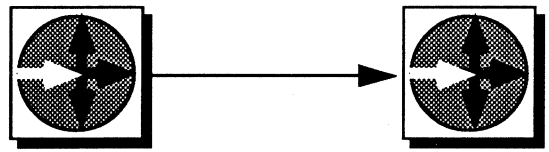
- Synchronous, Lower Performance
- Single Threaded
- Batched, Multiple Transactions
- High Speed LAN Preferred





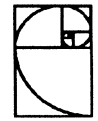
Replication Server To Replication Server Connections

- Efficient, Asynchronous Protocol
- Message Multiplexing
- Appropriate To Use Over WAN



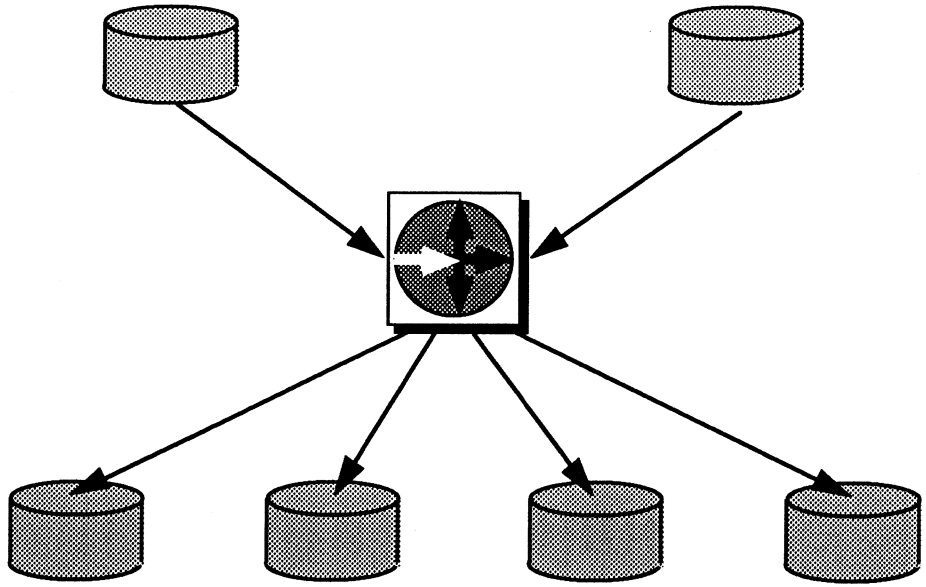
Don't Allow Data Server Logs To Fill Up!

- Failure To Drain Data Server Logs Will Prevent Their Truncation
- Disrupts Normal Transaction/Database Dump Maintenance Cycles.
- Primary Data Server And Replication Server Should Be Connected Via High Speed Reliable LAN And Co-located At Same Site Under Common Administrative Control.
- Replicate Data Servers Using Asynchronous Stored Procedures Should Also Be Co-located With Local Replication Server.



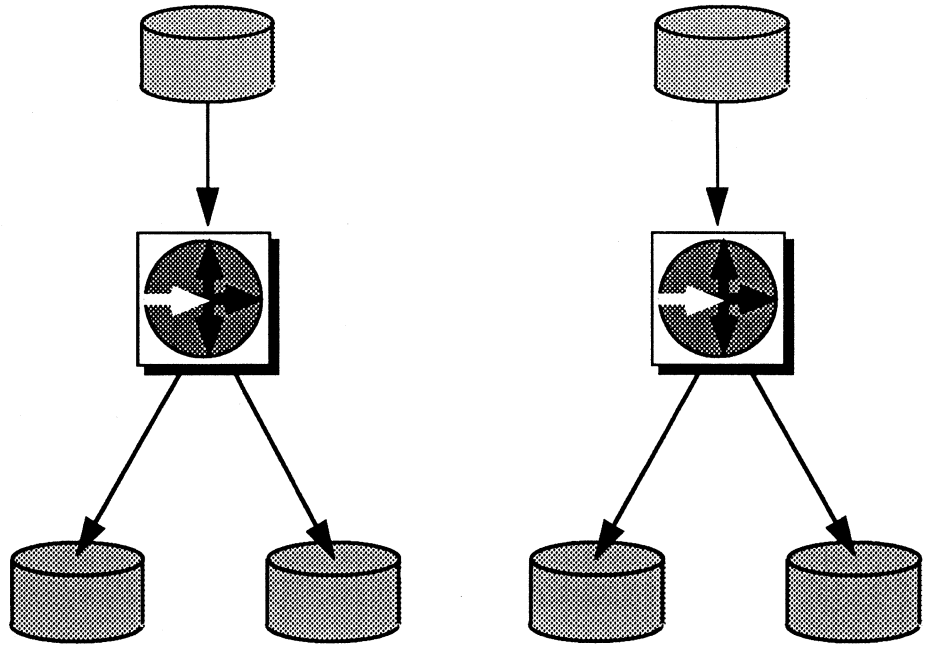
'Basic Design:'

- Single Replication Server
- Multiple Primary and Replicates
- Moderate Update Rate
- Single High Speed Reliable LAN



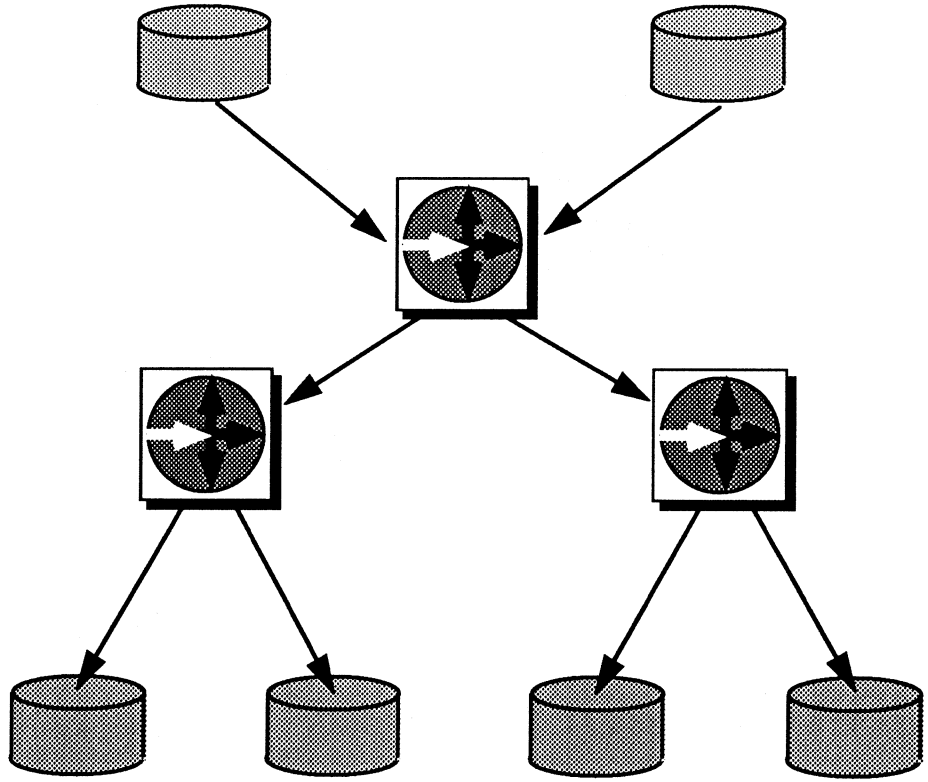
Adding Primary Replication Servers May Improve Performance If:

- There Are Too Many Primary Databases
- Combined Update Rate Is Too High



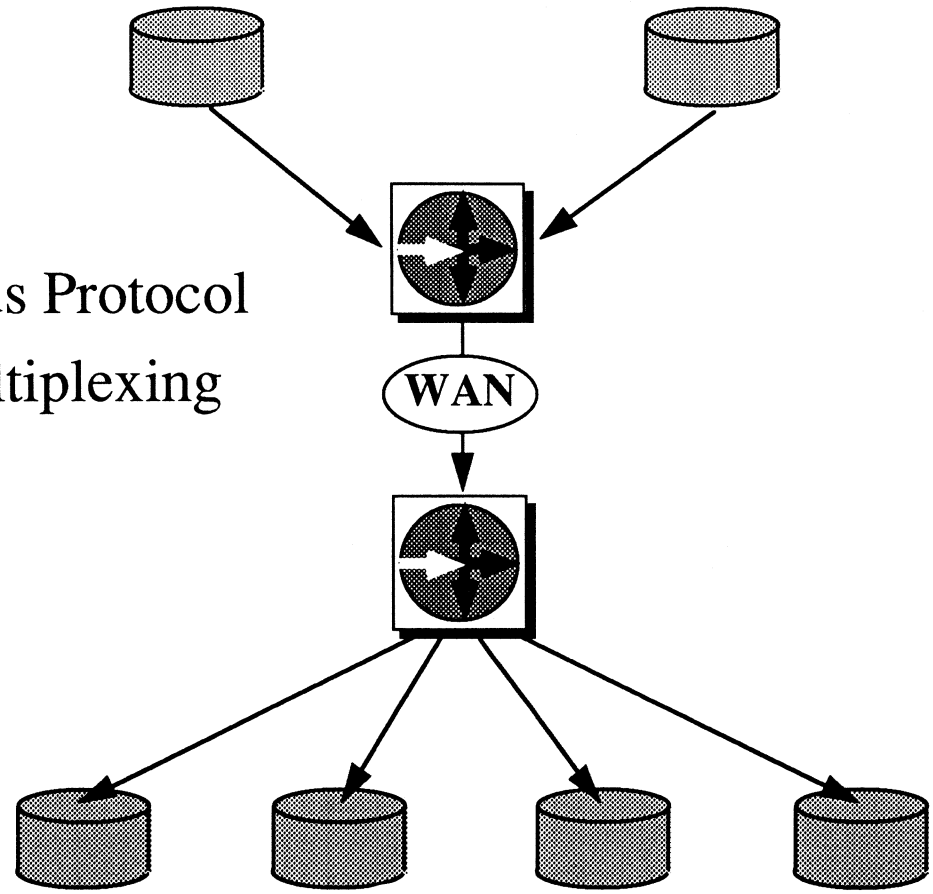
Adding Replicate Replication Servers May Improve Performance If:

- There Are Too Many Replicate Databases
- Combined Replicate Update Rate Is Too High



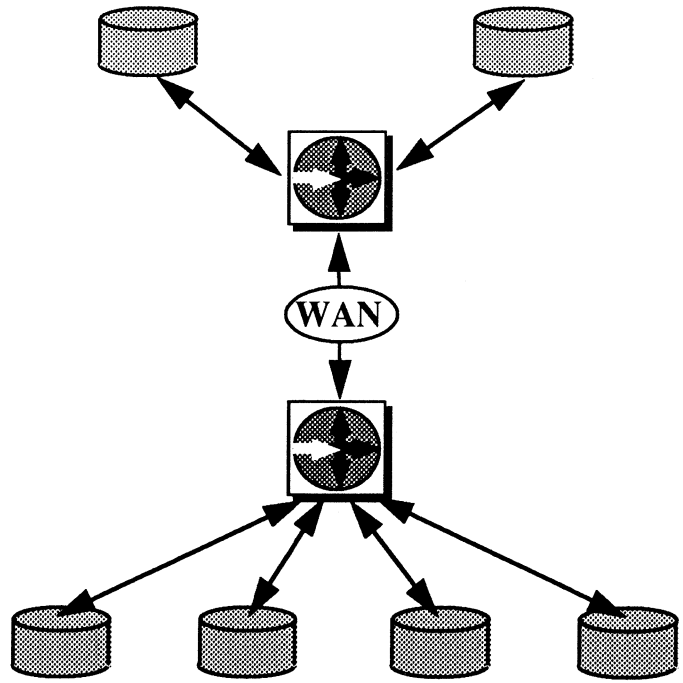
Use RS To RS Connections Over Slow WANS

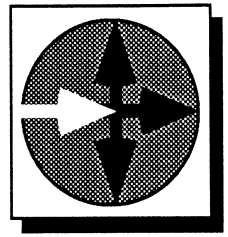
- Asynchronous Protocol
- Message Multiplexing



Collocate Replication and Data Servers:

- Co-locate Primary Data Server and Replication Server
- Co-locate Replicate Data Server and Replication Server When Asynchronous Stored Procedures are Used.

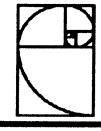




Replication Application Design Topics

Effects Of Loose Consistency

- System Failures May Delay The Propagation Of Data By Minutes, Hours, Or Days
- Under Some Conditions Users May Be Looking At Data That Is Very Old
- Under All Conditions, Data Will Lag The Primary By Some Interval
  - Sometimes As Little As A Few Seconds
  - Simple Utilities Can Be Written To Measure Latency (Heartbeat Utility)



Users Must Develop Guidelines Regarding Use Of Replication Vs. 2PC

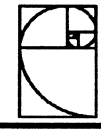
- To Book Travelers On Airlines?
  - Creates A Fast Reservation System, But -
  - Data Inconsistencies Can Leave People Without Seats, so -
  - Airline Can Compensate Passengers Or Roll Out Another Plane.
- To Trade Securities From The Same Inventory Or Position?
  - Users Must Be Prepared To Exercise Compensating Or Amending Transactions
  - However, Replication Allows Users To Amend Transactions Faster Than End Of Day Processing Would Allow Them To
- To Schedule Runways For Landing Aircraft?

Authoritative, Single Ownership Of Data

- Each Row Of Data Is 'Owned' By A Single Site
- Site Of Ownership Is Called 'Primary' With Respect To The Data Owned At The Site
- All Other Copies Are 'Replicates'
- All Updates Must Clear Through The Primary
- All Distributions Originate From Primary

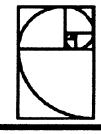
Updating Data

- Update Primary Synchronously
- Update Primary Asynchronously



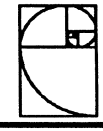
Updating Primary Synchronously

- Primary Must Be Available
- Communications Link To Primary Must Be Available
- Lower Performance And Availability
- Applications Access And Update Primary Database Directly
- Application Handles Transaction Failure Synchronously
- Concurrency Control Handled By Primary Data Server
- Appropriate To Use For 'High Value' Transactions



Updating Primary Data Asynchronously

- Primary Need Not Be Accessible
- Higher Application Performance, Availability
- Use Asynchronous Stored Procedures
- Applications May Need To Address Concurrency Control Issues
- Applications May Need To Address Transaction Failure Issues



What About Updating Replicate Data Directly?

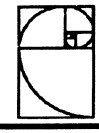
- Commonly Requested Feature
- Not Supported By Replication Server
- Not Recommended
- Often Indicates Improper Design (Replicate Should Be Primary, Or Application Must See Updates Immediately)
- Difficult Backup/Recovery Issues
- Commutative Updates May Make Sense:
  - Credit/Debit
  - Unique Inserts

Updating and Concurrency Control

- Synchronous: Handled By Primary Data Server
- Asynchronous: Three Techniques:
  - Avoidance:
    - Design Application To Avoid Need For Concurrency Control
    - All Transactions Are Unique
    - Updates Only Originate From One Site
    - Updates Only Originate From One Site At A Time
  - Compensation:
    - All Transactions Are Honored Regardless Of Conflict
    - Conflicting Transactions Trigger Compensating Transactions
  - Optimistic
    - Design Applications To Use Optimistic Concurrency Control
    - Use Version Number Or Timestamp
    - Application Verifies Timestamp On Each Update Request
    - Application Handles Transaction Re-submission When Conflict Is Detected

Handling Transaction Failures

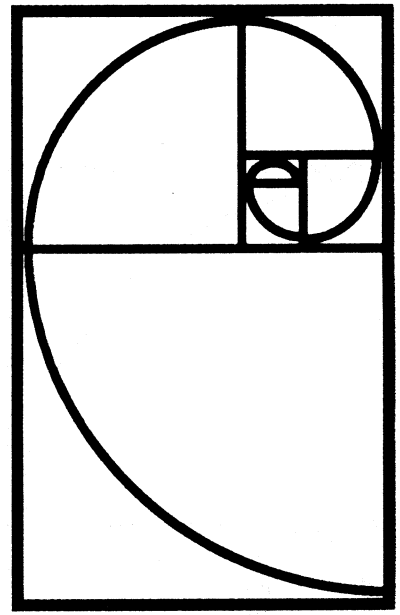
- Synchronous Access Is Handled Immediately By Applications
- Asynchronous Access:
  - Rollback Should Not Be Relied Upon For Expected Errors
  - Expected Error Conditions Should Be Handled Within Stored Procedures Without Rolling Back Transaction (Balance Overdraft, Concurrency Conflict..)
  - Unexpected Errors Should Cause System To Stop Until Error Is Corrected (Out Of Disc Space, Server Error,...)



Systems Management Presentation

Systems Management Products

Roadmap for Enterprise Client/Server Systems Management



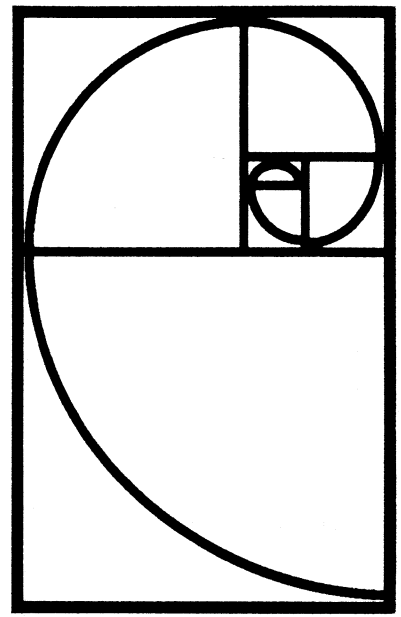
SYBASE

for Enterprise Client/Server Computing

By Steve Hershberg

Systems Management Products

Roadmap for Enterprise Client/Server Systems Management

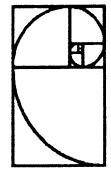


SYBASE

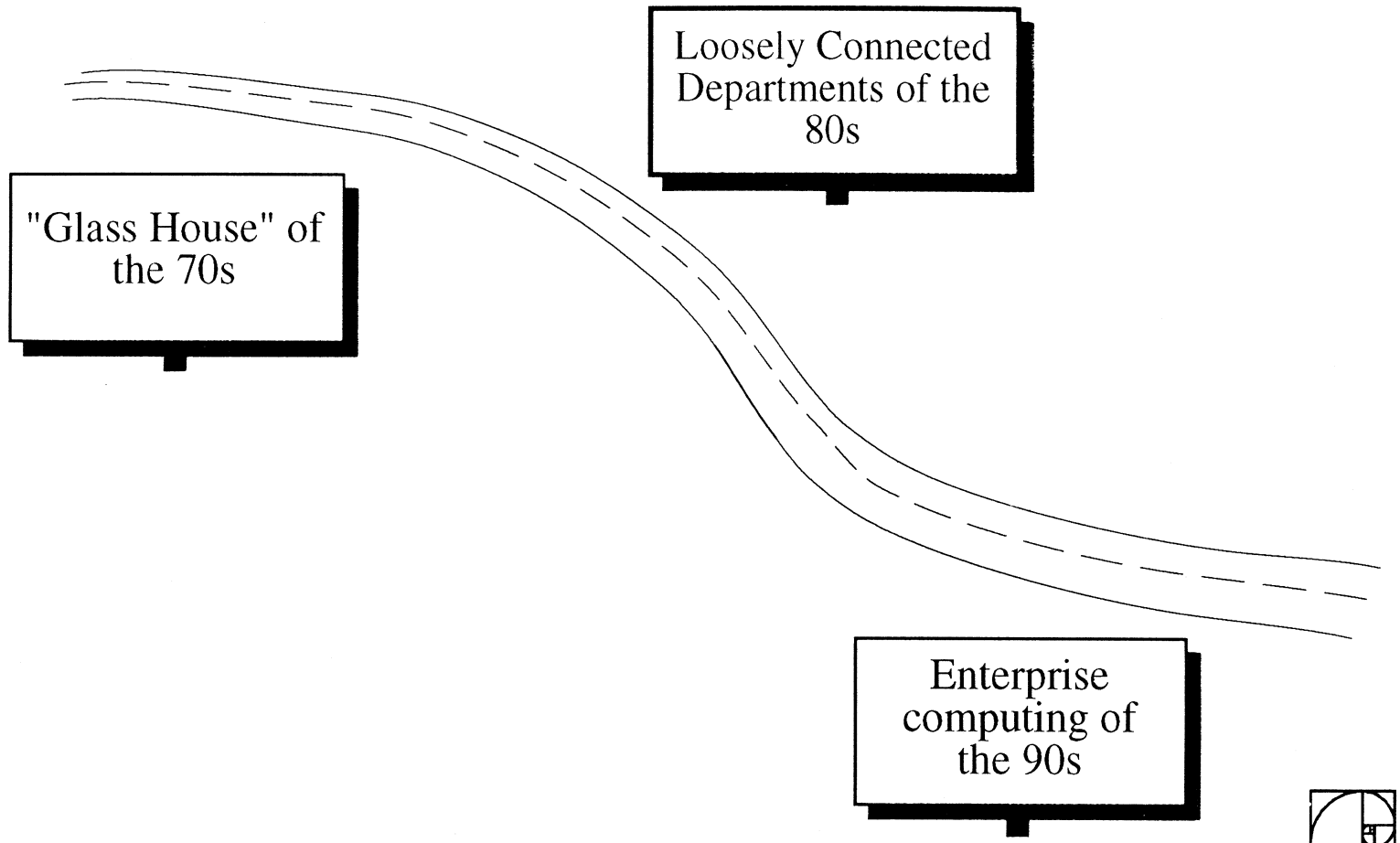
for Enterprise Client/Server Computing

Introduction

- Historical perspective
- Current toolset
- Future direction
- Product demonstrations



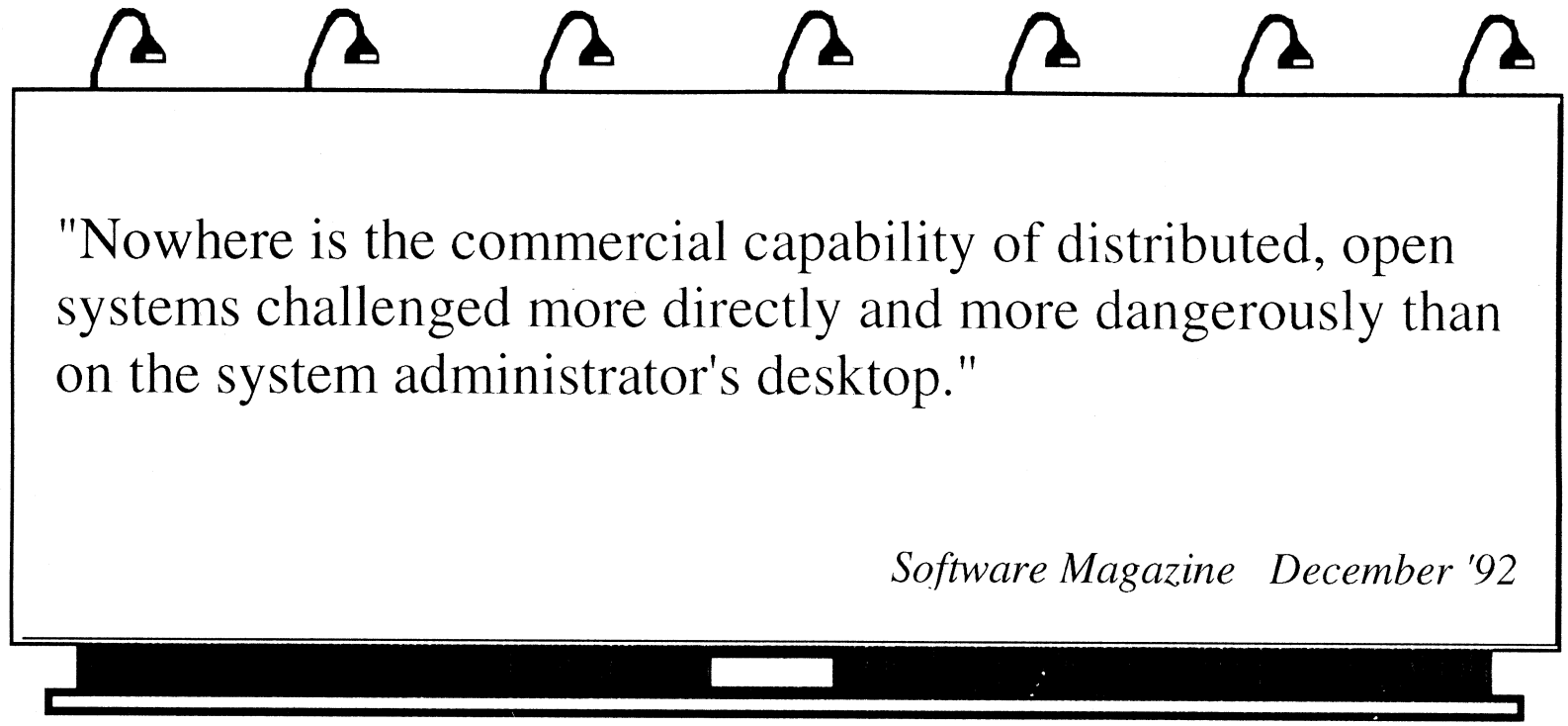
Roadmap to the 90s



Administration Concerns

- Storage
- Performance
- Security
- Integrity
- Control

The State of Systems Management



"Nowhere is the commercial capability of distributed, open systems challenged more directly and more dangerously than on the system administrator's desktop."

Software Magazine December '92

Systems Management Products

- **SA Companion**

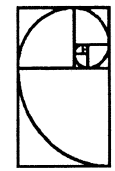
  Fully integrated operational control environment for SYBASE system administrators

- **SQL Monitor**

  Graphical performance monitoring tool specifically designed for the SQL Server

- **SQL Debug**

  GUI debugging tool for developing Transact-SQL applications



SA Companion Command Center

Server Management

- Connect
- Install
- Configure
- Examine error
- Recreate

Device Management

- List
- Add
- Drop
- Generate mirror

User Management

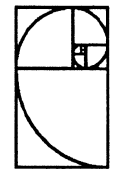
- List
- Add
- Delete
- Modify default

Reporting

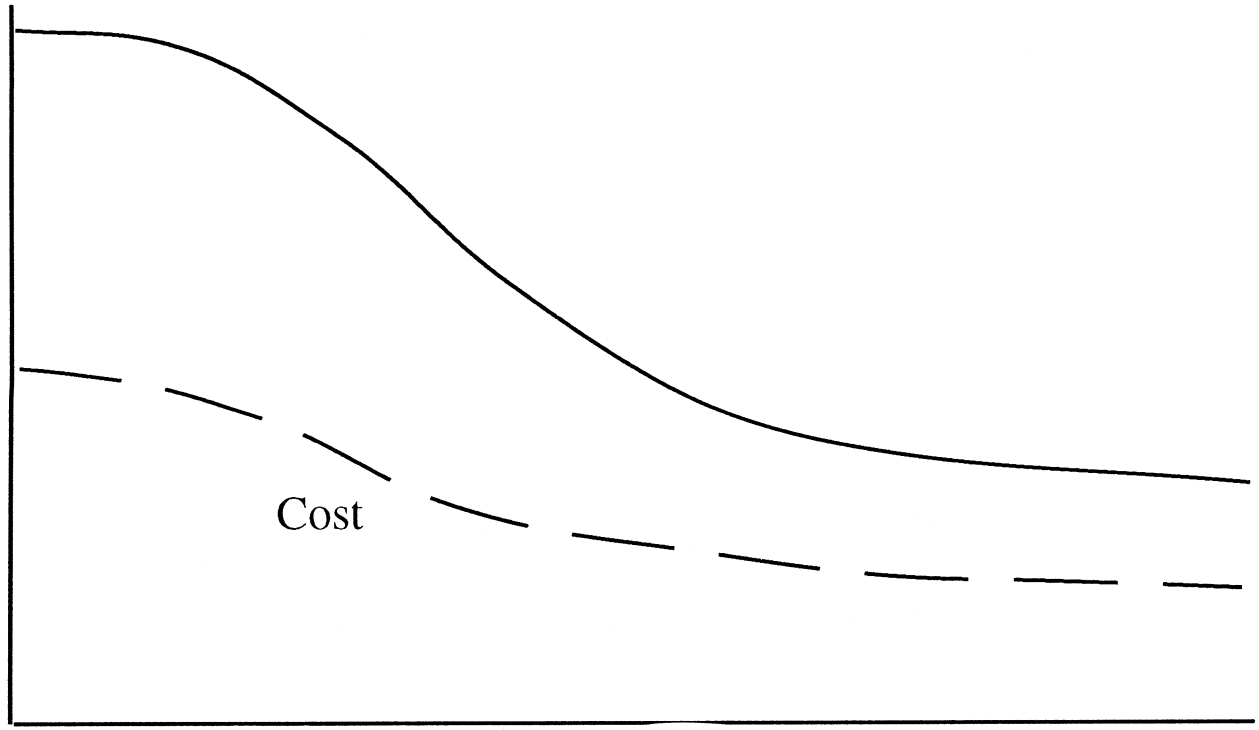
- Space Usage
- Server Logins
- Users
- Database Objects
- Groups

Database Management

- List
- Create
- Delete
- Estimate space
- Recreate

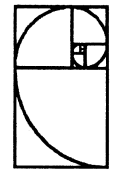


Administration Productivity

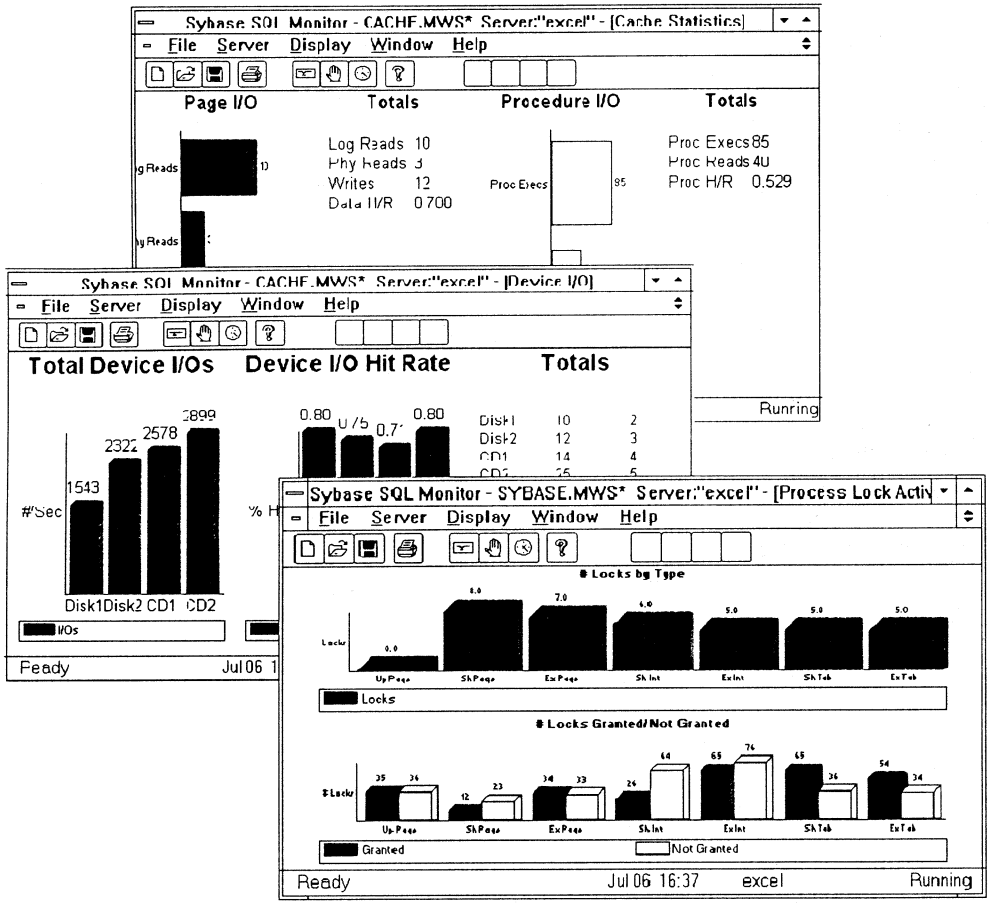


Efficient Use of System Resources

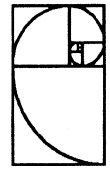
- Is my SQL Server configured properly?
- Are my database objects distributed for optimum performance?
- Where is my processing burden?
- Is there any application resource contention?



Monitoring SQL Server Performance

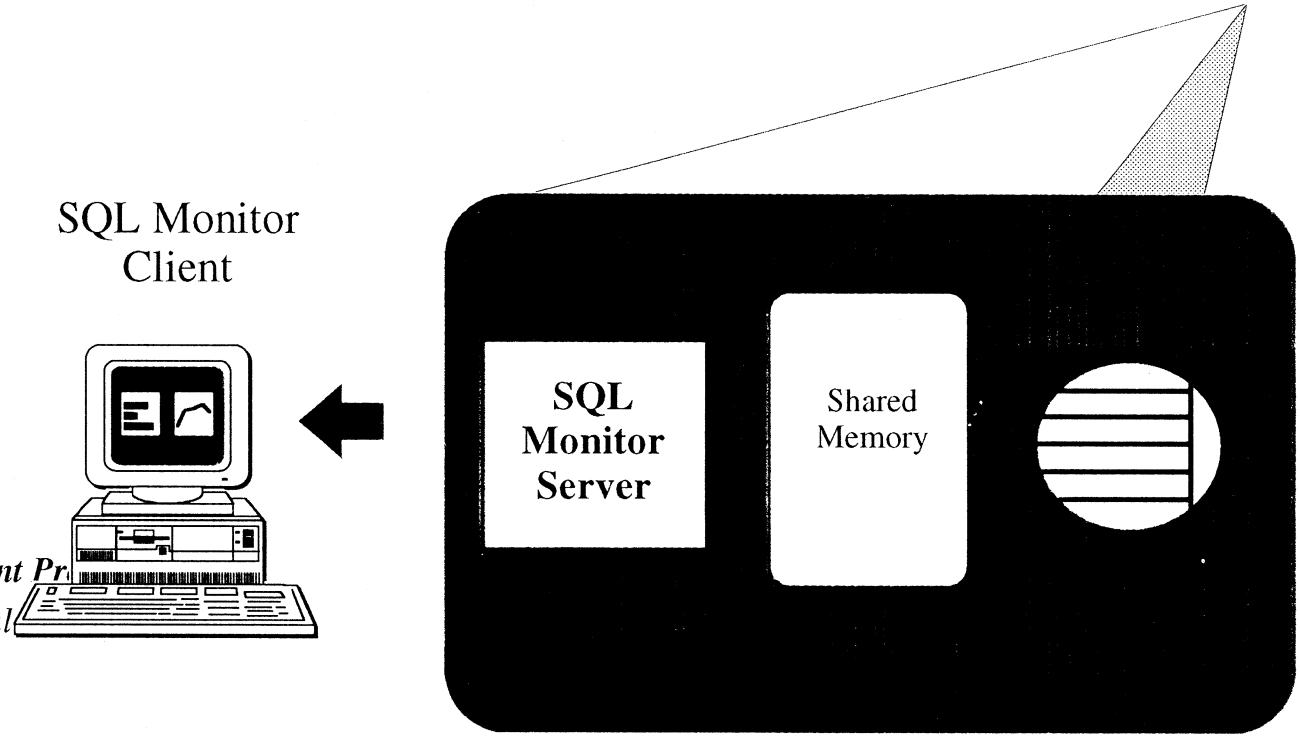


- Locks
- Cache
- Transaction activity
- Device I/O
- Process activity



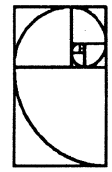
Monitor Without Processing Bur

SQL Monitor Architecture

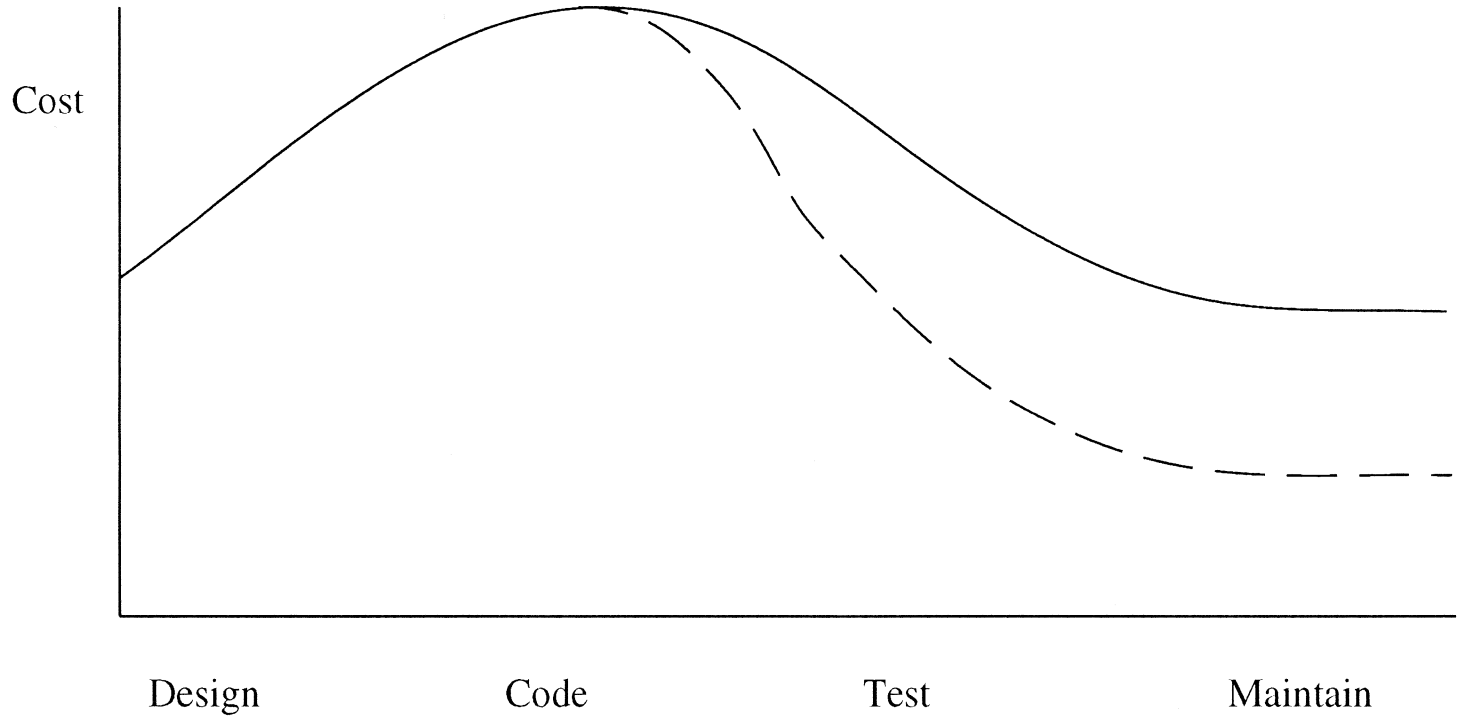


Transact-SQL Development Productivity

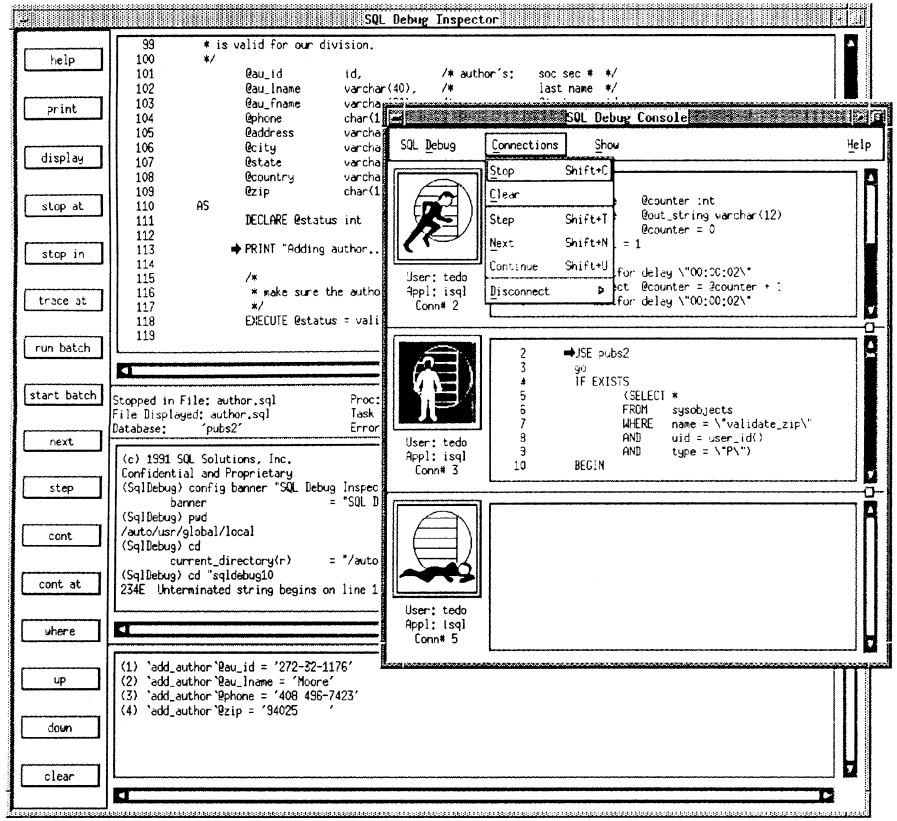
- Is the code thoroughly tested to ensure successful implementation?
- Is the code tuned for optimal performance?
- Are developers spending too much time trying to understand code written by somebody else?
- Has the locking logic been adequately tested?



System Development Lifecycle

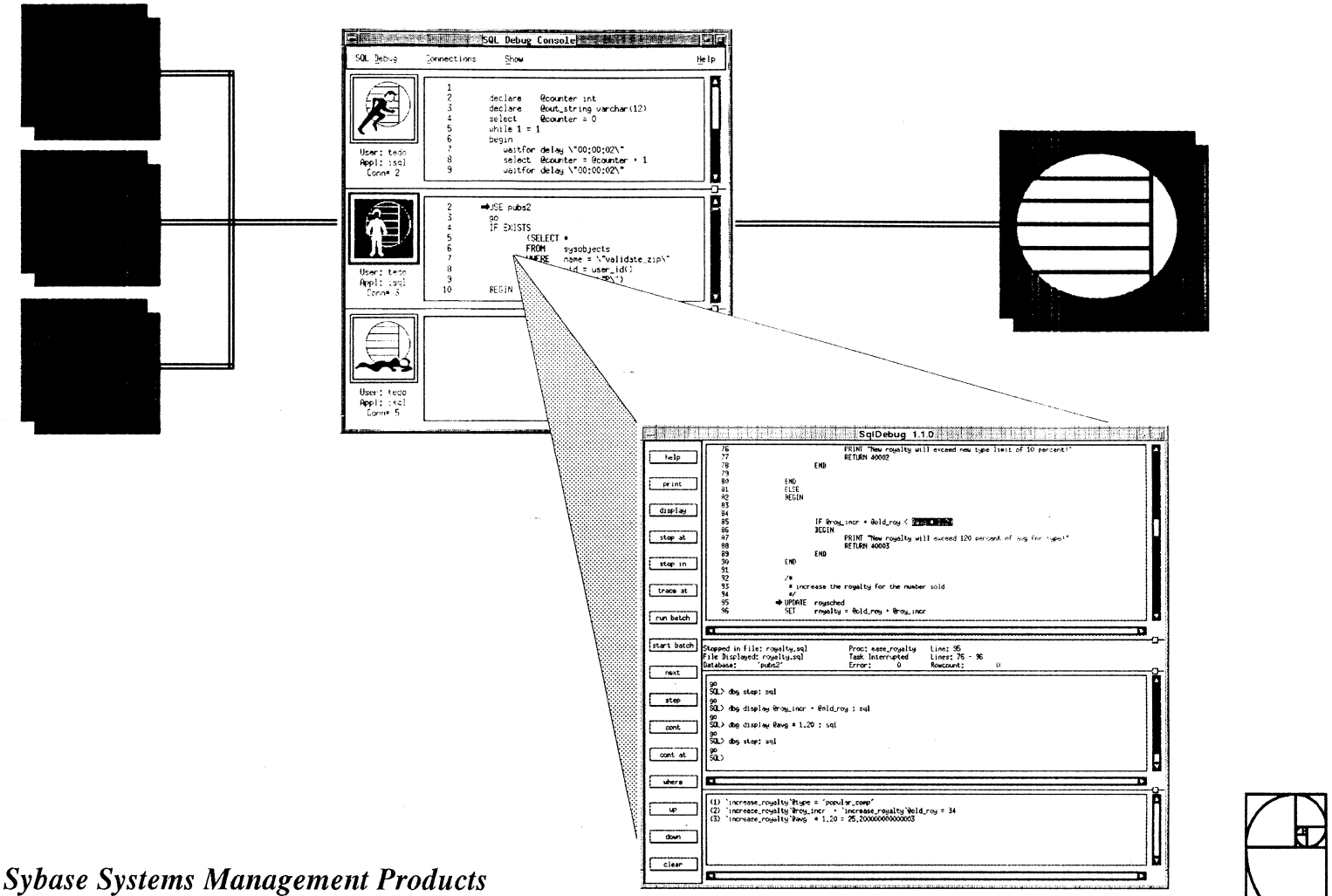


Essential for System Development



- Interactive Source Code Debugger
- Transact-SQL stored procedures
- Transparent debugging for Open clients

Debug any Open Client Application

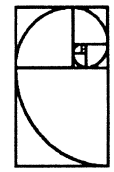


Sybase Systems Management Products

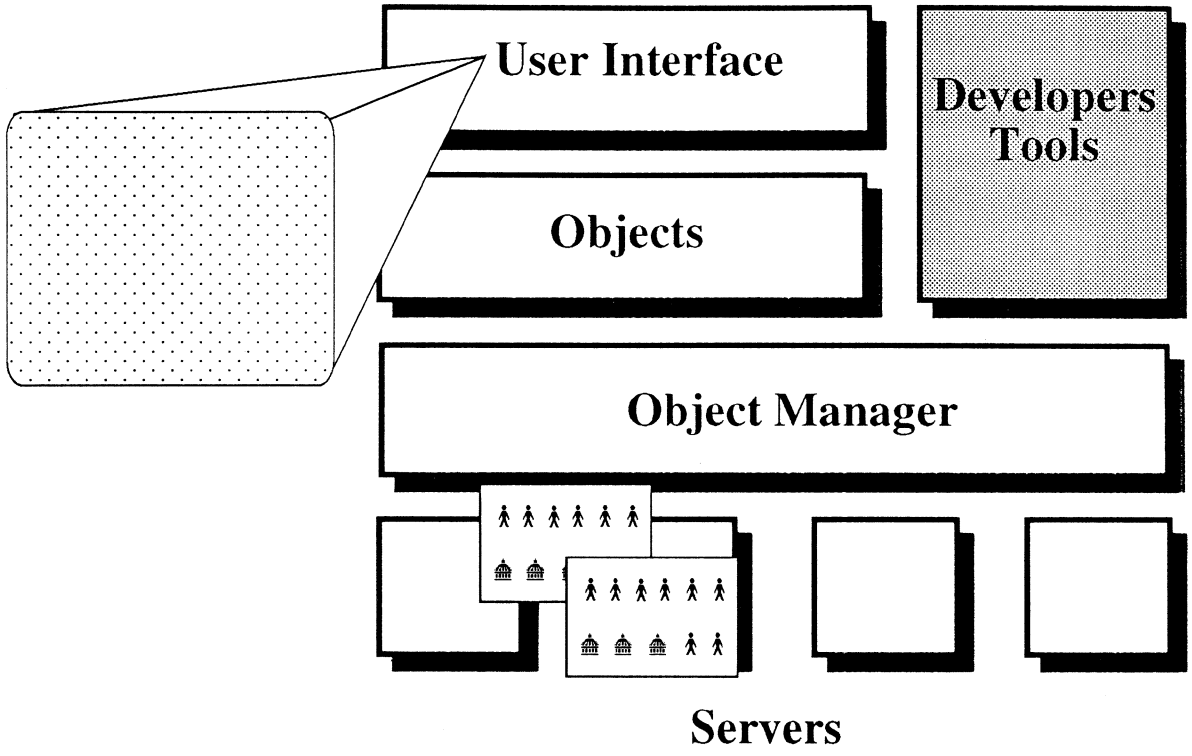
Proprietary and Confidential

SYBASE Systems Administration

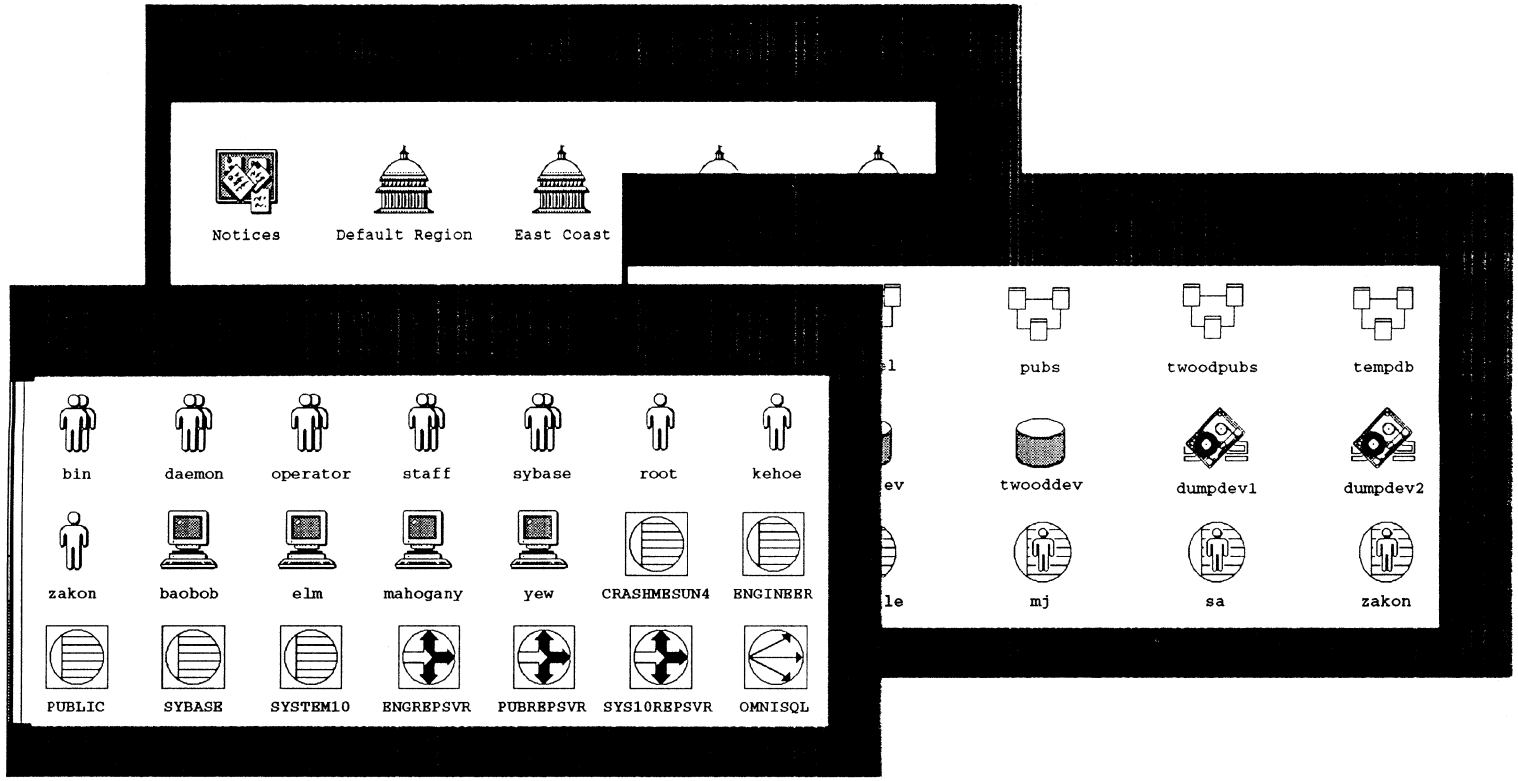
- Integrated systems management desktop with common interface
- GUI, Mac-like paradigm
- Interoperability - Standards-based solution (DME)
- Support for distributed environment
- Open, extensible tool



Framework for Enterprise Systems Administration (DME)



Systems Management Desktop



Sybase Systems Management Products

Proprietary and Confidential

Product Direction

- SQL Server administration and control
- SQL Server performance monitoring
- Replication Server administration
- OmniSQL Gateway administration
- Navigation Server administration