

Oracle 11i Tuning Advanced Pricing for Optimal Performance

An Oracle Technical White Paper
March 2001

Tuning Advanced Pricing for Optimal Performance

EXECUTIVE OVERVIEW

Oracle Advanced Pricing is designed to deliver maximum flexibility when pricing customer transactions. In and e-Business, transactions may be being entered by consumers using Oracle iStore, by telephone sales personnel using Oracle Tele-sales, from EDI orders being received electronically into Oracle Order Management, or from a variety of other sources. The applications mentioned all integrate with Advanced Pricing in Oracle Release 11i. All depend on the Advanced Pricing Engine to instantly and correctly price these transactions, regardless of the complexity of the pricing computations being performed.

Each time a customer transaction is entered, the Pricing Engine component of Advanced Pricing is called to search through the applicable pricing rules - called qualifiers - that apply to this transaction, and then to use these rules to select the correct set of pricing actions including price lists, formulas, discounts, and promotions that are needed to correctly price the transaction. Potentially, there are thousand or even tens of thousands of available actions that may have been setup in Advanced Pricing's internal tables, so the task of the Pricing Engine--providing maximum flexibility while delivering the extremely fast performance demanded by e-Businesses--is very challenging.

Throughout the development of Advanced Pricing, meeting the demanding performance requirements of e-Businesses has been a major design goal. To that end, the software design of the product has been optimized, and the Pricing Development team is continuously striving to find ways to further improve product performance, particularly of the pricing engine.

The choices you make when selecting certain settings, or how you elect to set up your pricing data can substantially improve the performance of the Advanced Pricing Engine.

The objective of this paper is to share with you our understanding and insights about these implementation and setup considerations, and to give you specific recommendations that if followed will improve the performance you receive from Advanced Pricing.

SOME TERMINOLOGY

Advanced Pricing utilizes pricing rules, called *qualifiers* to inform the pricing engine component of the software when to apply pricing actions to a transaction. There are several types of pricing actions, including *price lists*, *formulas*, and *modifiers*. Modifiers include such pricing actions such as giving a discount, promotional products, coupons, item substitutions, and several others.

The pricing engine cycles each time a *calling application* makes a pricing request to the engine. Calling applications in 11i include iStore, Order Management, Oracle Contracts, and Oracle Tele-sales.

PROCESS FLOW

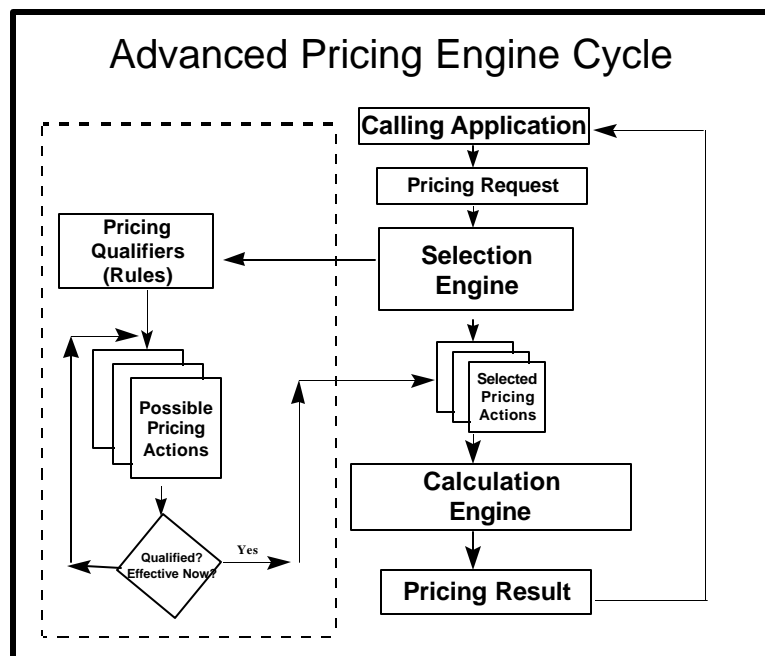


Figure 1 Pricing Engine Cycle Flow

Pricing Engine performance is the amount of cycle time that elapses between when the pricing request is submitted to the engine by the calling application, and when the pricing result is returned to the calling application by the engine.

The pricing engine goes through two types of processing activities each time it executes. First, the engine runs a selection engine component that evaluates the qualifier rules users have established, and based on those rules chooses the qualified 'pricing actions' that the calling application may need to apply the transaction they were processing when they called the pricing engine. Pricing actions include price lists, such formulas as may be attached to price lists, and a the wide variety of modifiers including discounts, promotional free goods, other

item discounts, accruals, and so forth.

The second activity is the calculation engine. The calculation engine processing begins when the selection engine has completed the task of selecting the proper pricing actions to apply to the transactions. The task of the calculation engine is to perform all necessary calculations needed to compute net selling prices.

Our development team's analysis of the Pricing engine's performance characteristics has revealed that the selection process is subject to more variability of execution time, because the number of records that may be selected is subject to much variability. Most of the material in this paper is aimed at improving selection engine performance.

ADVANCED PRICING SETUP CONSIDERATIONS

The best opportunity for improving Pricing engine performance can be had by analyzing the pricing data setups that the selection engine must process. In general, by purging unneeded qualifiers, price lists, and modifiers, you will be rewarded by improved Pricing Engine performance.

However, there are distributions of data in the pricing data setup that can slow the Pricing engine execution. In the remainder of this paper we will describe these, and make recommendations about how to avoid or eliminate them. The first of these we will deal with is qualifier selectivity.

Qualifier Selectivity

In the process flow topic presented previously, we learned that the selection component of the Pricing Engine looks at qualifiers to determine which pricing actions to apply. As the Pricing engine finds qualifiers that apply to a transaction, it preliminarily selects all price lists or modifier actions that the qualifier pertains to.

When pricing qualifiers are defined in such a manner that the qualifier identifies a narrow range of pricing actions, the majority of which should be applied to the transaction, then that qualifier has high selectivity. Conversely, when a single qualifier is linked to many pricing actions, the majority of which cannot be simultaneously applied to a transaction, then that qualifier is said to have low selectivity.

New Search Optimizer is introduced with the latest performance patch. Search optimizer introduces a mechanism by which pricing engine tags the most selective qualifier within a group of qualifiers attached to the same modifier. For example modifier "2% discount" has two qualifier namely Price list = Corporate and Customer = XYZ. Price List = Corporate is a non-selective qualifier because it is attached to many modifiers. However Customer = XYZ is a selective qualifier where its occurrence is very low. Pricing engine will first match the most selective qualifier within the qualifier group and then only match the less selective

qualifiers for the selected modifier. Search optimizer uses the number of occurrences as a criteria to tag the selectivity. Hence pricing engine would be intelligent to distinguish the selectivity of the qualifier “price list = common price list” and “price list = customer specific price list”.

If a modifier has qualifier as well as the product attached to it then the Pricing Engine is “Qualifier Driven” rather than “Product Driven”. For example a modifier “2% discount” has a qualifier attached “Price list = Corporate” and also has a product attached as “Item=ABC”. Pricing Engine will match the Qualifier first and then match the product. Hence it is extremely important that at least one qualifier is selective within the qualifier group.

Impact of Low Qualifier Selectivity

Low selectivity will have an adverse impact on Pricing engine performance. Whether the impact will be enough to be deleterious depends on the exact distribution of the pricing data. The larger the volume of data where qualifier selectivity is low, the greater the deleterious impact on performance.

Example of High Qualifier Selectivity

To illustrate this, let's look at a business examples involving promotional pricing, and consider how different pricing setups with high and low qualifier selectivity might be structured. Then we will look at the Pricing Engine performance implications of each.

Let's assume a hypothetical company whose customers belong to one of 3 groups: wholesale, retail, and other. This company normally gives a 2% discount throughout the year, but each quarter it also uses promotional pricing. When the company is having the promotion, it uses a special price list in lieu of the regular price list and it also gives promotional discounts that vary depending on the customer class. When the company is between promotions, it relies on a single price list, called 'Corporate' and offers one discount of 2%

Here is an example of pricing setup having high qualifier selectivity

Qualifier - Customer Class = 'Wholesale' Effective 2/15/2001 - 3/31/2001

Price List = First Quarter Wholesale Price List 2/15/2001 - 3/31/2001

Modifier = 5% Discount 2/15/2001 - 3/31/2001

in exclusivity group 1

Precedence = 100

Qualifier - Customer Class = 'Retail' Effective 2/15/2001 - 3/31/2001

Price List = First Qtr Retail Price List Effective 2/15/2001 - 3/31/2001

Modifier = 8% Discount

Precedence = 100

Qualifier - Customer Class = 'Other' Effective 2/15/2001 - 3/31/2001

Price List = First Qtr Other Price List Effective 2/15/2001 - 3/31/2001

Modifier = 3% Discount Effective 2/15/2001 - 3/31/2001

in exclusivity group 1

Precedence = 100

Price List = Corporate Price List 1/01/2001 - 12/31/2001

Modifier = 2% Discount Effective 1/01/2001 - 12/31/2001

in exclusivity group 1

Precedence = 200

In the above example, the pricing engine, when run, would find that a customer belongs to one of the three classes and then would use the price list and modifier for that specific class - Retail, Wholesale, or Other. It will also pull in the Corporate Price List and the 2 % discount modifier.

Since the price lists are by definition exclusive, and the modifiers have been assigned to exclusivity group 1, the engine will use precedence to select the price list and modifier that are qualified by customer class, because customer class is more specific than customer 'all'.

Performance of the engine using the above setup will be good, because the qualifiers cause the engine to retrieve price list and modifier records very selectively. In the cases where Price List is passed by the calling application (for example Order Management) qualifier will not be used to find the price list but it will be used only to validate the price list. Hence in such a situation qualifier selectivity may not impact the list price selection but will only impact Modifiers selection.

Example of Low Selectivity Due to Historical Records

Now lets take a case to illustrating lowered setup selectivity, where the cause is a large number of modifier and price list records with effectivity dates in the past.

Lets assume this company has been in business many years, and so has many price lists and discount records in their system, with many being outside their effectivity dates but still in the system for 'historical purposes'. Using our hypothetical company example, the setup data might look like this.

Qualifier - Customer Class = 'Wholesale' Effective 2/15/1991 -

Price List = First Quarter Wholesale Price List 2/15/2001 - 3/31/2001

= First Quarter Wholesale Price List 2/15/2000 - 3/31/2000

= First Quarter Wholesale Price List 2/15/1999 - 3/31/1999
and so forth back to 1991.

Modifier = 5% Discount 2/15/2001 - 3/31/2001
= 4% Discount 2/15/2001 - 3/31/2000
= 6.% Discount 2/15/2001 - 3/31/1999
and so forth back to 1991

in exclusivity group 1

Precedence = 100

Lets assume the other classes, Retail and Other, have data that looks largely the same. Now lets look at customer 'all', where management has experimented with many different discount structures over time.

Qualifier Customer All Effective 1/01/2001 - 12/31/2001

Price List = Corporate Price List 1/01/2001 - 12/31/2001

Modifier = 2% Discount Effective 1/01/2001 - 12/31/2001
= 1.8% Discount Effective 1/01/2000 - 06/30/2000
= 1.5% Discount Effective 7/01/2000- 12/31/2001
= 2% Discount Effective 1/01/2001 - 12/31/2001
and so forth back to 1991

in exclusivity group 1.

Precedence = 200

In the previous example, the qualifier selectivity is very low, negatively impacting pricing engine performance. When the pricing engine executes against this data, it will find that all the historical records (those with effectivity dates that are already past) will be preliminarily qualified and selected by the Pricing Engine for further processing, even though only one price list and modifier will be finally selected to apply to the transaction. Specifically, for the historical records, the pricing engine will be forced to compare the exterior pricing date passed to it from the calling application to the effectivity date range of each record to determine if whether the selection process should proceed to the next step of considering the precedence. Since the effectivity date evaluation is a record-by-record process, large number of historical records will have an adverse impact on Pricing engine performance.

Correcting Low Qualifier Selectivity due to 'Historical' Records

Clearly, an opportunity exists to improve Pricing Engine performance by properly handling 'historical' records. While the most direct route to improving engine

performance is to eliminate historical records from the system, many companies have a business process to need to retain historical records for a period of time.

Advanced Pricing provides a flag on both price list and modifier records that informs the engine whether the record is 'inactive' or 'active'. Since the Pricing Engine checks status of the record as part of the qualifier scan process, records set to 'inactive' are automatically excluded from being selected for further consideration.

Low Qualifier Selectivity Due to Release 10.7/11 Pricing

The predecessor releases of Oracle Applications provided far less flexibility about how you set up your pricing data. Advanced Pricing is much more flexible, providing you the capability to define your pricing rules and actions in a more concise manner than could be done in either Release 10.7 or Release 11.

In Release 10.7 and in Release 11.0, discounts had to be associated to a price list. Price List was a mandatory qualifier for a discount in R10.7/11. Because R10.7/R11 system functionality required users to create different discounts because in that release, one discount could only be linked to one price list. In turn, price lists could be linked to either the customer or the Order Entry order type, or they could be manually overridden on the order.

Therefore, in release 10.7 and 11.0, many Oracle customers find themselves with large numbers of discount records, even when the number of different discrete discounts in use is low.

In the upgrade for 11i, the relationship of qualifying the discount with the customer has of necessity been handled by for 11i by having these relationships be output into the 11i tables as. This can caused large number of qualifiers of type "price list", which will impact Pricing engine performance negatively.

Correcting Low Qualifier Selectivity due to Release 10.7/11 Upgrade

Here, as in the previous example, reducing of the number of discount records and making the qualifiers specific will help you optimize 11i Pricing Engine Performance. Consider merging these discount records into as few 11i modifier records as possible, and then using 11i Pricing capability to tie them to customer groups.

Very likely, if you examine your qualifiers and your business pricing requirement, you will find you can qualify pricing either price lists or modifiers at higher level of product hierarchy. For example if you are selling greeting cards and you only have 15 distinct prices but 100,000 items. By grouping the items together in to item categories, and using item category as the qualifier, you will have create only 15 price list lines rather than 100,000 lines. This will result in the Pricing engine searching through price list lines, which will boost Pricing engine performance

Also, if there is no business need for a Price list to act as a qualifier to a discount,

you should delete any such records that have been created as part of your upgrade processing.

Using Qualifiers as Constraints may result into Low Selectivity :

Example: If you have 1000 modifier lists. Out of which 200 lists have Price List = "Corporate" as the only qualifier . This results into Low selectivity of the qualifier because Pricing engine will be processing every list which satisfies this qualifier.

Correcting Low selectivity when Qualifiers is used as Constraints:

If your business requirement is to have a non selective qualifier as the only qualifier then consider combining these lists so that there will be less number of lists for engine to scan.

Redundant Qualifiers - Another Enemy of Performance

Performance of the Pricing engine can be boosted by avoid qualifiers that are redundant. Here is an example of a redundant qualifier:

Customer = 'XYZ' AND Customer Site ='ABC'

For purposes of the Pricing engine selecting the proper price list or modifiers, customer site is sufficiently specific to act as a qualifier, as it is more specific than customer. Adding customer qualifier causes the Pricing Engine to evaluate the customer condition unnecessarily. Eliminating such redundant qualifiers will act to speed the Pricing engine on its way.

Blind Modifiers

Modifiers without any qualifier or any product attached. These modifiers are processed by engine for every request line. Engine performance will be negatively affected as the more number of blind modifiers get defined in the system.

Use of "ALL_ITEMS" as a product

Modifiers defined for "ALL_ITEMS" are processed by the engine for every request line. Engine performance will be negatively affected as more number of modifiers having "ALL_ITEMS" get defined in the system.

USE OF EXCLUSIONS AND THE 'NOT=' OPERATOR

Please note that pricing engine will take additional processing time to evaluate "NOT=" operator in qualifiers as well as to evaluate "EXCLUDE" in the product hierarchy. If you have a very high volume of setup data then it is recommended that you use caution when implementing these operators.

ANALYZING YOUR DATA DISTRIBUTIONS: A SCRIPT

Pricing provides a script to analyze the data distribution. This script needs to be

run at the SQLPLUS prompt using APPS login and the results need to be provided to the pricing team in case of any performance bug logged by the customer.

TECHNICAL IMPROVMENTS TO PRICING ENGINE PERFORMANCE

When you implement Advanced Pricing, there are several technical measures that can be taken to ensure the best response time from the Pricing Engine. These measures are related to implementation time activities.

Attribute Mapping

New in Release 11i, Advanced Pricing provides the capability to perform attribute mapping. A patch is now available (please refer to Appendix 2) that allows you to use 'Static Generation' for attributes mapping. If in your implementation you are extending Advanced Pricing by mapping new attributes, then you will want to apply this patch, as it will give you for faster Pricing performance. Additionally, it also fixes a known PL/SQL memory issue.

Also, be careful while writing your own attribute sourcing. Please note that the code you will get write will be executed for every pricing engine call and, if not tightly written, negatively impact Pricing Engine performance.

Performance Opportunities with Phases and Events

11i Advanced Pricing provides a configurable capability that allows the pricing engine execution to be broken up into phases, and allows each phase to be associated with an event taking place in the calling application. This presents some implementation time opportunities to improve Pricing engine performance.

If there is no need to fetch or view the discounts at the time of entering the order lines, then you can modify the event phases records to do execute the discounting phases just at the line save. This approach will cause the engine to perform the price list line selections as each line is entered, while preventing the engine from doing the selection or calculation of modifiers until the line save event causes the modifier selection to cycle. For users not needed to view the discounts line by line as the ordered is entered, this technique can enhance perceived pricing engine performance.

Another recommended idea, is, in the case of unused Pricing Event phases, to set End date for an unused phases to a date in the past. The effect of this is to prevent Pricing Engine from attempting selection activity when the event that triggers the Pricing Phase occurs.

Temporary Tablespace

Advanced Pricing makes extensive use of the temporary tablespace feature of Oracle 8i. Therefore, proper sizing of temporary tables is very important to obtaining optimal performance. Depending on the size of the transaction that will

be priced, the initial extent for temporary tables should be sized at between 64K and 256K.

Additionally, the temporary tablespace should be defined as locally managed.

Memory Considerations:

Since Pricing engine is frequently called during order entry process it is important that the pricing packages are always loaded into the memory. It is recommended that you PIN e.g. keep the following Pricing packages in the memory

1. QP_PREQ_GRP
2. QP_BUILD_SOURCING_PVT
3. QP_resolve_incompatability_PVT
4. QP_FORMULA_PRICE_CALC_PVT
5. QP_Calculate_Price_PUB
6. QP_CUSTOM

IMPORTANT! MAKE SURE THAT YOUR SHARED_POOL SIZE IS APPROPRIATELY CALCULATED BASED ON THE USE OF THE SYSTEM.

PERFORMANCE IN THE PRICING SETUP SCREEN

Following performance related improvements in the setup screen have been made:

- Price List setup screen: Query List price by product - A new find window has been provided
- Agreements Setup screen - A new find window has been provided
- Modifiers Screen - A List of Values Customer Name, site_use, ship_to has been provided

A word of caution: The user is responsible for the performance of the List of Values/ validation of the user extended qualifiers/pricing attributes. Hence please make sure that the where clause in the user defined Value set is properly tuned.

PERFORMANCE IN UPGRADE FROM 10.7/11 TO 11.I

Performance in upgrading the 10.7/11 pricing has been improved by

- providing parallel threads
- providing bifurcation so that the pricelists/modifiers of active orders can be upgraded first and other data upgrade can be deferred to a later convenient time.

A Suggestion! Purging the unneeded pricing data prior to the upgrade will improve the performance of the pricing upgrade as well as improving the performance of the pricing engine. While purging the data please ensure that the data integrity of the transaction system is maintained. An approach some customers have successfully used for accomplishing this is to purge the price list lines of the obsolete price list and upgrade just the price list header to maintain the data integrity.

PERFORMANCE IN APPLYING CERTAIN PRICING DE-NORMALIZATION PATCHES

If you have a very high volume of price list and modifiers data then you may experience that the certain pricing patches take anywhere from 5 minutes up to an hour to apply. The reason for this delay is that the patch also includes de-normalization script for the existing data. Please do not kill the application of this patch. These de-normalized columns are important for the pricing engine to select the list price or modifiers. Large de-normalization patches have been provided with parallel threading to improve the performance.

CONCLUSION

This technical white paper has given an overview of the major causes of poor Pricing Engine performance, and has outlined corrective measures. For additional information on Advanced Pricing, see the Oracle Advanced Pricing User's Guide.

APPENDIX 1: PERFORMANCE PATCHES (SUBJECT TO CHANGE)

Following patches have been provided to improve the Performance of the Pricing Engine. *You should always check with support for any changes in this list of patches.*

Patch #	Pack B	Pack C	Pack D	11i.3	Comments
Pricing					
1426285	X				Oct 2000
1520579	X	X		X	Nov 2000
1508982	X	X		X	Dec 2000
1555985	X	X		X	Dec 2000
1620189	X	X	X	X	Dec 2000
1531826	X	X			Nov 2000
1545351	X	X	X	X	Mar 2001

The above table lists the pricing patch numbers. Check mark (X) indicates that the patch needs to be applied if you are at that patch set level.

Important:! Certain performance patches have been released in the area of Pricing and OM Integration. These patches are under the product “Order Management”. Please check with your support representative to obtain these patches.

APPENDIX 2: DATA DISTRIBUTION ANALYSIS SCRIPT

Important! *This script is subject to change. Always check with Oracle Support to ensure you have latest version of this script.*

```
column qualifier_context format a10
column qualifier_attribute format a22
column qualifier_attr_value format a15
column qualifier_attr_value_to format a10
column active_flag format a5 head 'ACTIV'
set pages 66
set lines 80
set feed on
title 'Pricing Data Distribution'
spool qp_perf_distr.lis
prompt 'QP_List_Headers'
prompt '===== '
select list_type_code, active_flag, count(*) from qp_list_headers_b
group by
active_flag, list_type_code;
prompt 'QP_List_Lines'
prompt '===== '
prompt 'QP_List_lines - distribution by type, qual, phase'
select list_line_type_code, qualification_ind , pricing_phase_id ,
count(*) from
qp_list_lines group by list_line_type_code, qualification_ind,
pricing_phase_id ;
prompt 'QP_List_lines - distribution by modifier_level_code '
select modifier_level_code,
count(*) from
qp_list_lines group by modifier_level_code;
prompt 'QP_List_lines - end dated modifiers '
select count(*)
      from qp_list_lines a, qp_list_headers_b b
where a.list_header_id = b.list_header_id and
b.list_type_code not in('PRL','AGR') and b.active_flag = 'Y'
and a.end_date_active is not null ;
prompt 'QP_PRICING_ATTRIBUTES'
prompt '===== '
prompt 'QP_Pricing_Attributes- grouping by phase '
select pricing_phase_id, count(*) from qp_pricing_attributes group by
pricing_phase_id ;
prompt 'QP_QUALIFIERS'
prompt '===== '
prompt 'QP_Qualifiers- Total Count'
prompt '===== '
```

```

SELECT count(*) from qp_qualifiers;
prompt 'QP_Qualifiers- distinct qualifiers '
prompt '===== '
SELECT qualifier_context, qualifier_attribute, count(*) from
qp_qualifiers
group by qualifier_context, qualifier_attribute;
prompt 'QP_Qualifiers- Highest non-selective '
prompt '-===== '
select QUALIFIER_CONTEXT,QUALIFIER_ATTRIBUTE,
QUALIFIER_ATTR_VALUE,
QUALIFIER_ATTR_VALUE_TO , count(*) from qp_qualifiers where
active_flag = 'Y' group by
QUALIFIER_CONTEXT,QUALIFIER_ATTRIBUTE,
QUALIFIER_ATTR_VALUE,
QUALIFIER_ATTR_VALUE_TO having count(*) > 50;
prompt 'QP_Qualifiers- header vs line '
prompt '===== '
select count(*) from qp_qualifiers where list_line_id is not null;
prompt 'QP_Qualifiers- grouping by operator '
select count(*),COMPARISON_OPERATOR_CODE from
qp_qualifiers group by
COMPARISON_OPERATOR_CODE ;
prompt 'QP_rltd_modifiers'
prompt '===== '
prompt 'QP_rltd_modifiers- group Count'
SELECT RLTD_MODIFIER_GRP_TYPE, count(*) from
qp_rltd_modifiers
group by RLTD_MODIFIER_GRP_TYPE;
prompt 'END of script'
prompt '*****'
exit
/

```



Tuning Advanced Pricing to Ensure Optimal Performance
March 2001

Author: Tony Maxey

Contributing Authors: Nitin Hase, Jayrama Holla

Oracle Corporation

World Headquarters

500 Oracle Parkway

Redwood Shores, CA 94065

U.S.A.

Worldwide Inquiries:

Phone: +1.650.506.7000

Fax: +1.650.506.7200

Web: www.oracle.com

This document is provided for informational purposes only and the information herein is subject to change without notice. Please report any errors herein to Oracle Corporation. Oracle Corporation does not provide any warranties covering and specifically disclaims any liability in connection with this document.

Oracle is a registered trademark, and Oracle Order Management is (are) a trademark(s) or registered trademark(s) of Oracle corporation. All other names may be trademarks of their respective owners.

Copyright © Oracle Corporation 2001

All Rights Reserved