Fantrap Support in Discoverer Version 3.1.41

An Oracle Technical White Paper

April 2000

ORACLE®
Enabling the Information Age™

Fantrap Support in Discoverer Version 3.1.41

Introduction

This white paper will discuss the well known fantrap phenomenon in query and reporting environments. It will present a scaleable (i.e. server based) and maintenance free solution for this problem. Until now administrators of query and reporting tools in general were struggling with implementing maintenance intensive workarounds and very complicated solutions in order to provide end users with correct results. The Oracle Discoverer development team has put a lot of effort in finding a final solution to this problem. The solution is presented in this white paper.

Statement of the problem

Two basic architectures (OLTP systems and data warehouses)

The phenomenon 'fantrap' typically appears in situations where information is retrieved from relational databases via SQL in which multiple tables are involved. In general there are two basic architectures in which the fantrap occurs. The first situation is the so-called master-detail-detail (MDD) model, typically found in OLTP systems. In this model the first detail table is the master for the second detail table. An example of the MDD model is shown in figure 1. CUSTOMER is the master table, SALE is the first detail table having a second detail table SALE_DETAILS.

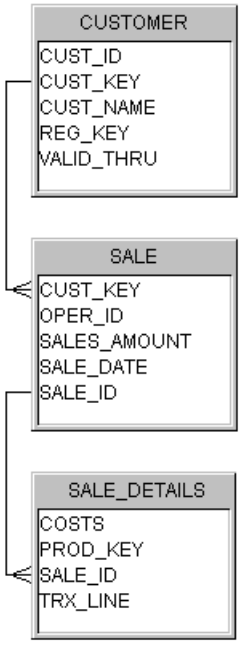

Figure 1. *(Right) An example of a MDD model*

The second situation is found in data warehouses with multiple fact tables and common dimension tables. In data warehouse environments multiple-fact-table data models are not unusual. According to data warehouse experts like Ralph Kimball, a fact table is subject oriented and is dimensioned by a set of dimension tables. In a data warehouse environment multiple subjects may exist and therefore it is not unusual that a data warehouse model contains more than one fact table with common dimensions. This model is indicated as MF2 (an acronym for master-fact1-fact2). This model is shown in figure 2.

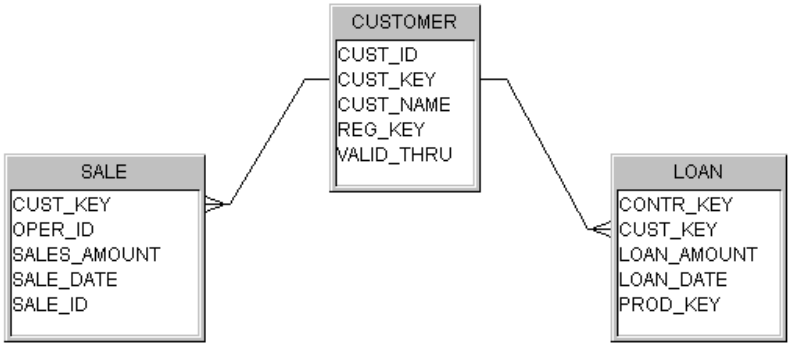

Figure 2. *(Left) An example of a MF2 model*

Both systems are used as data source by end users for information retrieval using ad-hoc query and analysis tools like Oracle Discoverer. These tools use dynamically generated SQL to retrieve data from these data sources based on an intermediate semantic layer. The semantic layer is the business area in the EUL for Discoverer. The following section explains why the basic models are potential sources of erroneous result sets for ad-hoc SQL generating query tools.

Why does this go wrong in straight SQL ?

The semantic layer in query tools hide database complexity for end users. Basically the database tables and columns are presented in terms of commonly known and accepted objects. When end users create an ad-hoc report these objects or items are selected. The semantic layer generates the corresponding SQL by selecting the corresponding tables, join paths and applied constraints. In fact, the basic components of a SQL statement (SELECT, FROM, WHERE, GROUP BY, etc.) are filled in.

Let us see what happens when a report is created based on the basic architectures. The examples used in this paper are based on the data model given in Appendix A.

Example 1: The MDD basic model

Based on the presented MDD model a report can be created showing the COSTS and SALES_AMOUNT per CUSTOMER. Following the strategy described above, the SQL statement generated is:

```
SELECT M.CUST_NAME         CUSTOMER
,      SUM(D.SALES_AMOUNT) SALES
,      SUM(DD.COSTS)               COSTS
FROM   CUSTOMER     M
,      SALE         D
,      SALE_DETAILS DD
WHERE  M.CUST_KEY=D.CUST_KEY
AND    D.SALE_ID=DD.SALE_ID
GROUP BY M.CUST_NAME
```

which results into:

```
CUSTOMER   SALES COSTS
-------- ------- -----
CUMMINGS  800,00  3.10
GRANT    1400,00  7.45
JOHNSON  1000,00  2.00
MCCLOUD   300,00  0.30
OSBORNE  4800,00  5.60
```

This is very wrong and should be

```
CUSTOMER   SALES COSTS
-------- ------- -----
CUMMINGS  800,00  3.10
GRANT     900,00  7.45
JOHNSON  1000,00  2.00
MCCLOUD   300,00  0.30
OSBORNE  1600,00  5.60
```

The SALES column contains wrong results because some sales records have more than one detail record, i.e. SALE_ID=12 and SALE_ID=16, which is confirmed by the temporary result of the SQL statement above before the aggregation and GROUP BY:

```
CUSTOMER SALE_ID SALES COSTS
-------- ------- ----- -----
JOHNSON       11  1000  2.00
GRANT         12   500  2.50
GRANT         12   500  0.75
GRANT         13   400  4.20
CUMMINGS      14   800  3.10
MCCLOUD       15   300  0.30
OSBORNE       16  1600  1.10
OSBORNE       16  1600  0.50
OSBORNE       16  1600  4.00
```

Example 2: The MF2 basic model

Based on the presented MF2 model a report can be created showing the SALES_AMOUNT and LOAN_AMOUNT per CUSTOMER. Straight SQL would result into:

```
SELECT C.CUST_NAME         CUSTOMER,
       SUM(S.SALES_AMOUNT) SALES_REVENUE,
       SUM(L.LOAN_AMOUNT)  LOAN_REVENUE
FROM   CUSTOMER            C
,      SALE                S
,      LOAN                L
```

```
WHERE  C.CUST_KEY = S.CUST_KEY
AND    C.CUST_KEY = L.CUST_KEY
GROUP BY C.CUST_NAME
```

Which results into:

CUSTOMER	SALES_REVENUE	LOAN_REVENUE
CUMMINGS	2400	160
GRANT	900	20
JOHNSON	2000	140
MCCLOUD	600	30
OSBORNE	1600	40

And is wrong again. It should be:

CUSTOMER	SALES_REVENUE	LOAN_REVENUE
CUMMINGS	800	160
GRANT	900	10
JOHNSON	1000	140
MCCLOUD	300	30
OSBORNE	1600	40

If for a given customer, the number of sales transactions differs from the number of loan transactions, the result is wrong for that specific customer, which is confirmed by the temporary result:

CUSTOMER	SALES_REVENUE	LOAN_REVENUE
CUMMINGS	800	30
CUMMINGS	800	30
CUMMINGS	800	100
GRANT	400	10
GRANT	500	10
JOHNSON	1000	50
JOHNSON	1000	90
MCCLOUD	300	20
MCCLOUD	300	10
OSBORNE	1600	40

In fact the join created in the SQL statement above is not valid. There is no semantic relationship between sale and loan transactions. Therefore the query should be solved in two separate queries and printed next to each other in the report.

In fact both data warehouse and OLTP data models may contain these basic models. MDD models appear in OLTP schemas and MF2 in data warehouse models for which multiple fact tables are implemented. The next section shows some solutions implemented by query and reporting tools.

Known solutions

SQL generating query and reporting tools in the Business Intelligence market place have been struggling for years to solve these kinds of problems (these tools are sometimes called the *desktop* OLAP tools).

In all versions up to patch release 3.1.40, Oracle Discoverer has been able to detect a potential fan trap situation. The detection works properly for the MF2 situation only (master with multiple detail tables) and is based on the foreign key relationships defined in the database schema or defined in the Business Area by the Oracle Discoverer Administrator (see figure 3)

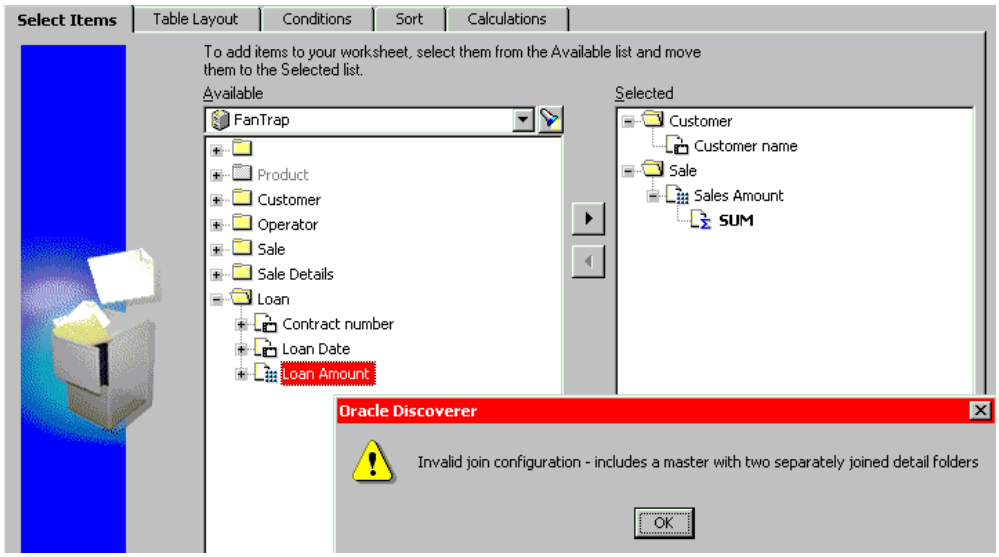

Figure 3. *Fantrap detection in Oracle Discoverer until release 3.1.40*

When the detection is disabled the user may obtain wrong results. When the fan trap detection is enabled the user will not be able to create a report based on two detail tables, unless the Discoverer Administrator creates a UNION ALL custom folder based on a union of both fact tables. However this solution is not satisfactory in a situation with very large fact (detail) tables, i.e. it is not a scaleable solution. It is not a manageable solution either with many fact tables: the Discoverer Administrator has to create a custom folder for all possible combinations of fact tables.

Fantrap support in Oracle Discoverer

This section will discuss the solution of Oracle Discoverer to the fantrap problem. It will show that this solution is generic for all data models. The solution will be presented based on both basic models .

The concept of inline views

The solution implemented in Oracle Discoverer version 3.1.41 is server based and involves the use of inline views. In the previous section the SQL statements without inline views retrieved wrong results because the tables involved are joined before the single-group function (aggregation, SUM) is performed. In fact, each table involved in the query should be grouped first before it is joined with a related table. The foreign key columns of the aggregated table, required to join with a related table, together with referenced primary keys, constitute the GROUP BY clause of the query. This concept is the basis for the algorithm implemented in Discoverer. Let us work out this concept with the basic model examples.

Example 1: The MDD basic model

The required information is CUSTOMER.CUST_NAME, SALE.SALES_AMOUNT and SALE_DETAILS.COSTS.

The tables involved in this example are:

TABLE_NAME	MEASURES	PRIMARY KEY COLUMN(S)	FOREIGN KEYS COLUMN(S)
CUSTOMER	-	CUST_KEY	-
SALE	SALES_AMOUNT	SALE_ID	CUST_KEY
SALE_DETAILS	COSTS	TRX_LINE	SALE_ID

Table 1. *Tables involved in example 1, MDD. The right column contains the relevant foreign key columns. Primary keys columns in yellow are referenced by related foreign keys.*

The CUSTOMER table does not store any measures (data points). The SALE table contains a measure to which a single-group function must be applied: SALES_AMOUNT. The foreign key to CUSTOMER is CUST_KEY, referencing the primary key of CUSTOMER, i.e. CUST_KEY. The SALE_DETAILS table also contains a measure: COSTS. The foreign key to SALE is SALE_ID referencing the primary key of SALE, i.e. SALE_ID.

First for each of the two detail tables, an inline view is defined. In this example we will call them SALE_IVW and SALE_DETAILS_IVW respectively.

Inline view for SALE:

```
(                                                  CUST_KEY    SALE_ID SALES_AMOUNT
SELECT S.CUST_KEY          CUST_KEY                ---------- ---------- ------------
,      S.SALE_ID           SALE_ID                          1         12          500
,      SUM(S.SALES_AMOUNT) SALES_AMOUNT                     1         13          400
FROM   SALE                S                                2         14          800
GROUP BY S.CUST_KEY, S.SALE_ID                              3         15          300
) SALE_IVW                                                  4         11         1000
                                                            5         16         1600
```

Inline view for SALE_DETAILS:

```
(
SELECT D.SALE_ID           SALE_ID                   SALE_ID      COSTS
,      SUM(D.COSTS)        COSTS                   ---------- ----------
FROM   SALE_DETAILS        D                               11       2.00
GROUP BY D.SALE_ID                                         12       3.25
) SALE_DETAILS_IVW                                         13       4.20
                                                           14       3.10
```

```
        15       0.30                                    16       5.60
```

Consequently each in-line is used to constitute the final SQL statement:

```
SELECT M.CUST_NAME         CUSTOMER
,      SUM(D.SALES_AMOUNT) SALES
,      SUM(DD.COSTS)              COSTS
FROM   CUSTOMER            M
,      SALE_IVW            D
,      SALE_DETAIL_IVW     DD
WHERE  M.CUST_KEY = D.CUST_KEY
AND    D.SALE_ID = DD.SALE_ID
GROUP BY M.CUST_NAME
```

The same analysis holds for the second example, the MF2 basic model:

Example 2: The MF2 basic model

The required information is CUSTOMER.CUST_NAME, SALE.SALES_AMOUNT and LOAN.LOAN_AMOUNT.

The tables involved in this example are:

TABLE_NAME	MEASURES	PRIMARY KEY COLUMN(S)	FOREIGN KEYS COLUMN(S)
CUSTOMER	-	CUST_KEY	
SALE	SALES_AMOUNT	SALE_ID	CUST_KEY
LOAN	LOAN_AMOUNT	-	CUST_KEY

Table 2. *Tables involved in example 2, MF2. The right column contains the relevant foreign key columns. Primary keys columns in yellow are referenced by related foreign keys.*

The CUSTOMER table, again, does not store any measures. The SALES table and LOAN table contain a measure to which a single-group function must be applied: SALES_AMOUNT and LOAN_AMOUNT respectively. The foreign key to customer is CUST_KEY, referencing the primary key of CUSTOMER, i.e. CUST_KEY for both tables.

According to the inline view concept together with table 2 the following in-line views are defined, called SALE_IVW and LOAN_IVW respectively in the example printed below.

Inline view for SALE:

```
(
SELECT C.CUST_KEY          CUST_KEY
,      SUM(S.SALES_AMOUNT) SALES_AMOUNT
FROM   CUSTOMER            C
,      SALE                S
WHERE  C.CUST_KEY = S.CUST_KEY
GROUP BY C.CUST_KEY
)      SALE_IVW
```

```
  CUST_KEY SALES_AMOUNT
---------- ------------
         1          900
         2          800
         3          300
         4         1000
         5         1600
```

Inline view for LOAN:

```
(
SELECT C.CUST_KEY          CUST_KEY
,      SUM(L.LOAN_AMOUNT)  LOAN_AMOUNT
FROM   CUSTOMER     C
,      LOAN         L
WHERE  C.CUST_KEY = L.CUST_KEY
GROUP BY C.CUST_KEY
)      LOAN_IVW
```

```
  CUST_KEY  LOAN_AMOUNT
----------  -----------
         1           10
         2          160
         3           30
         4          140
         5           40
```

Consequently each in-line is used to constitute the final SQL statement:

```
SELECT M.CUST_NAME                CUSTOMER
,      SUM(D1.SALES_AMOUNT)       SALES
,      SUM(D2.LOAN_AMOUNT)        LOAN
FROM   CUSTOMER     M
,      SALE_IVW     D1
,      LOAN_IVW     D2
WHERE  M.CUST_KEY = D1.CUST_KEY
AND    M.CUST_KEY = D2.CUST_KEY
GROUP BY M.CUST_NAME
```

From version 8.1.5 of Oracle database, inline views are not only supported in the FROM clause but also in the SELECT clause. This opportunity enables us to use in-line views in a more compact and effective way.

```
SELECT C.CUST_NAME  CUSTOMER
,      (
       SELECT SUM(S.SALES_AMOUNT)
       FROM   SALE          S
       WHERE  S.CUST_KEY = C.CUST_KEY
       )      SALES_REVENUE
,      (
       SELECT SUM(L.LOAN_AMOUNT)
       FROM   LOAN          L
       WHERE  L.CUST_KEY = C.CUST_KEY
       )      LOAN_REVENUE
FROM   CUSTOMER     C
```

This alternative but very elegant SQL statement, based on in-line views in the SELECT clause, reflects the fact that there is no relationship at all between sale and loan transactions. *(We would like to thank Lex de Haan who analyzed this basic model and brought the solution in line with relational calculus principles).*

Oracle Discoverer version 3.1.41 implements both solutions mentioned. Figure 4 shows the generated SQL in the MF2 example.

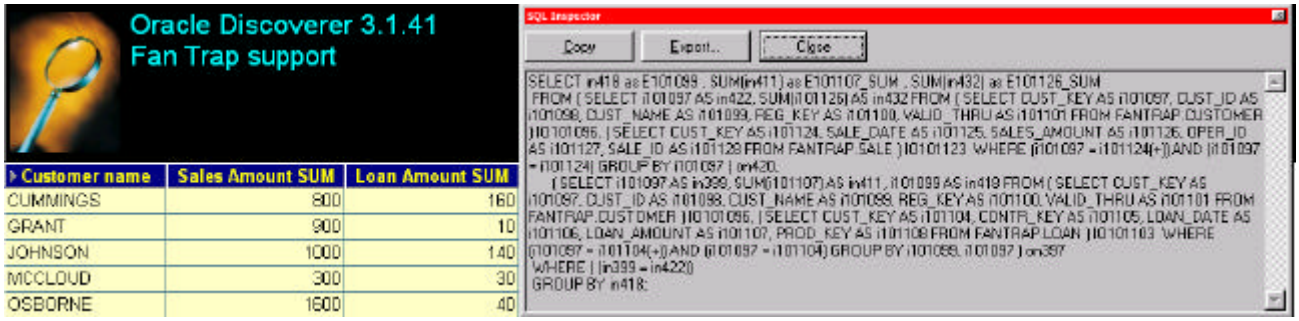

Figure 4. *Generated SQL for the MF2 example.*

Performance, scaleability and maintenance

The solution described above is a server based solution because the query is performed entirely by the database. The intelligence of recognizing a fantrap situation is in the Discoverer EUL. The temporary inline views are created on the server and are joined together on the server:

The solution is completely maintenance free. This is especially important with frequently changing data models. During the setup and maintenance of the Business Area the Discoverer Administrator does not have to reckon with any additional settings for possible fantrap situations. The only aspect that is important for the generation of correct SQL are constraints. The Discoverer Administrator must setup the table constraints properly either in the database or in the Business Area. If the database changes a refresh of the Business Area is sufficient to reflect these changes.

This architecture ensures the solution to be very scaleable, maintenance free, client machine independent and gives optimal performance.

Concluding remarks

This document presented the most important enhancement of Oracle Discoverer version, 3.1.41, which is fantrap support. Fantrap situations commonly occur in small and large data warehouse environments. It has been shown that the solution implemented by Oracle Discoverer is easy to understand, free of maintenance and very scaleable. The technical, server based architecture of the fantrap support implementation guarantees optimal performance and does not rely on any additional knowledge of end user and administrators concerning fantrap environments.

Appendix A. Demo Table Descriptions

This appendix describes the demo schema and tables used throughout this document.

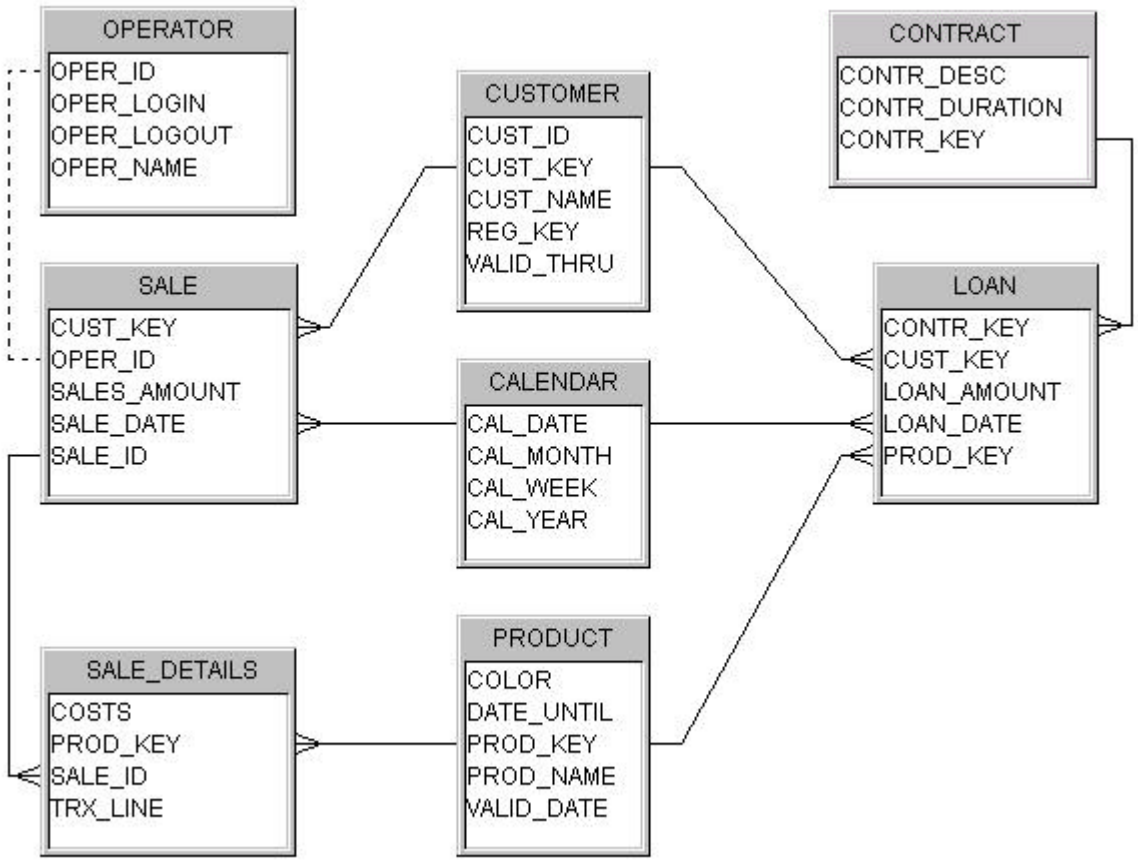

Figure 1. *The demo data model*

Description and contents of the tables are shown on the next pages.

Description of the demo tables. Primary keys are indicated in ***blue, bold italics***, foreign keys in blue only.

TABLE: OPERATOR

NAAM	TYPE
OPER_ID	***NUMBER(3)***
OPER_NAME	VARCHAR2(10)
OPER_LOGIN	DATE
OPER_LOGOUT	DATE

TABLE: CUSTOMER

NAAM	TYPE
CUST_KEY	***NUMBER(10)***
CUST_ID	NUMBER(10)
CUST_NAME	VARCHAR2(50)
REG_KEY	NUMBER(10)
VALID_THRU	DATE

TABLE: CALENDAR

NAAM	TYPE
CAL_DATE	***DATE***
CAL_WEEK	NUMBER(2)
CAL_MONTH	NUMBER(2)
CAL_YEAR	NUMBER(4)

TABLE: PRODUCT

NAAM	TYPE
PROD_KEY	***NUMBER(10)***
PROD_NAME	VARCHAR2(50)
COLOR	VARCHAR2(10)
VALID_DATE	DATE
DATE_UNTIL	DATE

TABLE: CONTRACT

NAAM	TYPE
CONTR_KEY	***NUMBER(1)***
CONTR_DESC	VARCHAR2(20)
CONTR_DURATION	VARCHAR2(10)

TABLE: SALE

NAAM	TYPE
CUST_KEY	NUMBER(10)
SALE_DATE	DATE
SALES_AMOUNT	NUMBER(10)
OPER_ID	NUMBER(3)
SALE_ID	***NUMBER(2)***

TABLE: LOAN

NAAM	TYPE
CUST_KEY	NUMBER(10)
CONTR_KEY	NUMBER(10)
LOAN_DATE	DATE
LOAN_AMOUNT	NUMBER(10)
PROD_KEY	NUMBER(10)

TABLE: SALE_DETAILS

NAAM	TYPE
TRX_LINE	***NUMBER(3)***
SALE_ID	NUMBER(2)
PROD_KEY	NUMBER(10)
COSTS	NUMBER

TABLE: OPERATOR

OPER_ID	OPER_NAME	LOGIN	LOGOUT
2	JOHN	14-01-1999 20:00:00	14-01-1999 22:00:00
1	PETER	17-01-1999 09:00:00	17-01-1999 11:30:00
2	JOHN	24-02-1999 09:00:00	24-02-1999 10:00:00
1	PETER	03-03-1999 14:00:00	03-03-1999 14:30:00
1	PETER	01-05-1999 12:00:00	01-05-1999 17:00:00

TABLE: CUSTOMER

CUST_KEY	CUST_ID	CUST_NAME	REG_KEY	VALID_TH
1	1001	GRANT	1	01-01-99
2	1002	CUMMINGS	2	01-01-99
3	1003	MCCLOUD	2	01-01-99
4	1004	JOHNSON	2	01-01-99
5	1002	CUMMINGS	1	01-03-99

TABLE: PRODUCT

PROD_KEY	PROD_NAME	COLOR	VALID_DATE	DATE_UNTIL
1	Sparta Standard	BLACK	01-01-00	10-01-00
2	Sparta Race	RED	01-01-00	10-01-00
3	Gazelle Oldtimer	SILVER	01-01-00	10-01-00
4	Gazelle Tandem	GREEN	01-01-00	10-01-00
5	Raleigh Sprint	BLUE	10-01-00	14-01-00

TABLE: CONTRACT

CONTR_KEY	CONTR_DESC	CONTR_DURATION
1	LONG TERM CONTRACT	1 YEAR
2	MIDDLE TERM CONTRACT	30 DAYS
3	SHORT TERM CONTRACT	7 DAYS
4	ONE DAY CONTRACT	24 HOURS

TABLE: SALE

CUST_KEY	SALE_DATE	SALES_AMOUNT	OPER_ID	SALE_ID
4	14-01-99	1000	2	11
1	14-01-99	500	2	12
1	17-01-99	400	1	13
2	24-02-99	800	2	14
3	03-03-99	300	1	15
5	01-05-99	1600	1	16

TABLE: SALE_DETAILS

TRX_LINE	SALE_ID	PROD_KEY	COSTS
111	11	1	2.00
112	12	1	2.50
113	12	3	0.75
114	13	5	4.20
120	14	4	3.10
121	15	2	0.30
130	16	3	1.10
131	16	4	0.50
132	16	5	4.00

TABLE: LOAN

CUST_KEY	CONTR_KEY	LOAN_DATE	LOAN_AMOUNT	PROD_KEY
2	3	02-01-99	100	2
1	4	04-01-99	10	1
3	1	30-01-99	10	3
4	3	01-02-99	50	2
2	1	02-02-99	30	3
2	4	20-02-99	30	1
5	2	10-03-99	40	3
3	3	11-03-99	20	1
4	1	03-04-99	90	3

TABLE: CALENDAR

Records for 01-jan-1999 to 31-12-2000

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
+1.650.506.7000
Fax +1.650.506.7200
http://www.oracle.com/