SQL an PL/SQL New features in Oracle 12c R2

laszlo.czinkoczki@webvalto.hu

Siófok, 2019, HOUG 2019

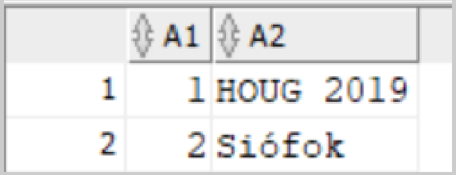
SQL New Features

- CREATE TABLE Enhancements
  - Using sequencies in the table definition (explicitly)
  - Using identity in the table definition
    Using sequencies in the table definition (implicitly)
- Adaptive Query Optimization
- CREATE VIEW Enhancements
- SELECT Enhancements
- Using PL/SQL subprograms in SQL Statements
- Adaptive Plans
- New or Enhanced Functions
- Creating an using Analytic Views

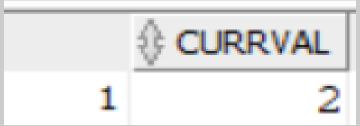
Using sequence in CREATE TABLE statement

A sequence can be used to generate values for PK and UK

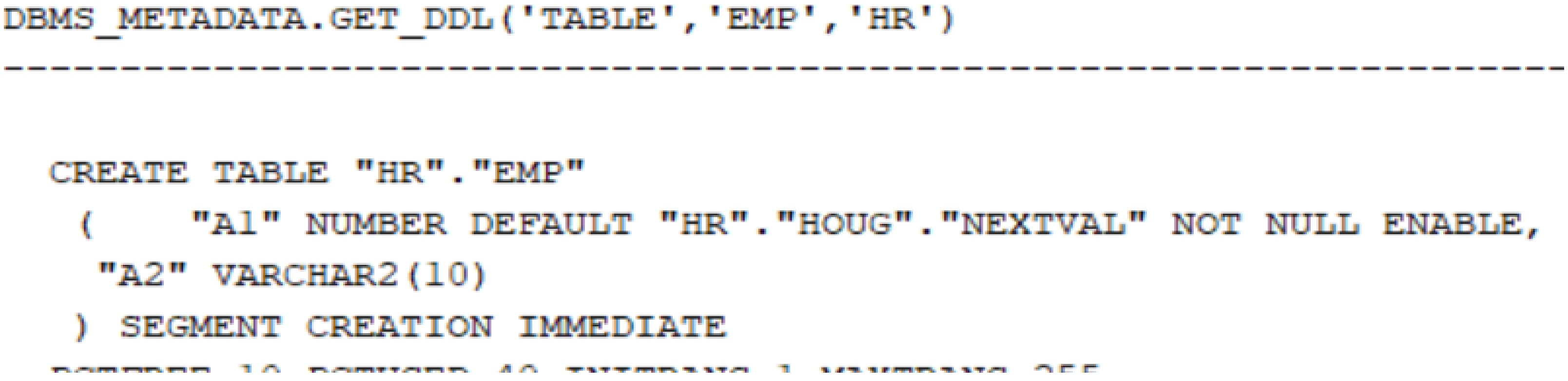
```
DROP SEQUENCE HOUG;
DROP TABLE EMP PURGE;
CREATE SEQUENCE HOUG START WITH 1;
CREATE TABLE emp
(a1 NUMBER DEFAULT HOUG.NEXTVAL NOT NULL,  a2 VARCHAR2(10));
INSERT INTO emp (a2) VALUES ('HOUG 2019');
INSERT INTO emp (a2) VALUES ('Siófok');
COMMIT;
SELECT * FROM emp;
SELECT  houg.CURRVAL  FROM dual;
SELECT DBMS_METADATA.GET_DDL('TABLE','EMP','HR')  FROM DUAL;
```



	A1	A2
1	1	HOUG 2019
2	2	Siófok



	CURRVAL
1	2



```
DBMS_METADATA.GET_DDL('TABLE','EMP','HR')
--------------------------------------------------------------------------------


  CREATE TABLE "HR"."EMP"
   (    "A1" NUMBER DEFAULT "HR"."HOUG"."NEXTVAL" NOT NULL ENABLE,
    "A2" VARCHAR2(10)
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
```

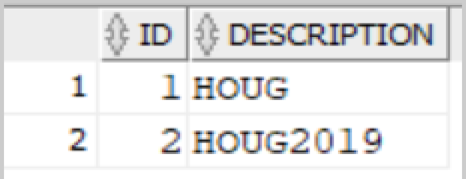
Using identity in CREATE TABLE statement

- You create an identity column.
- Oracle will create and use a sequence automatically

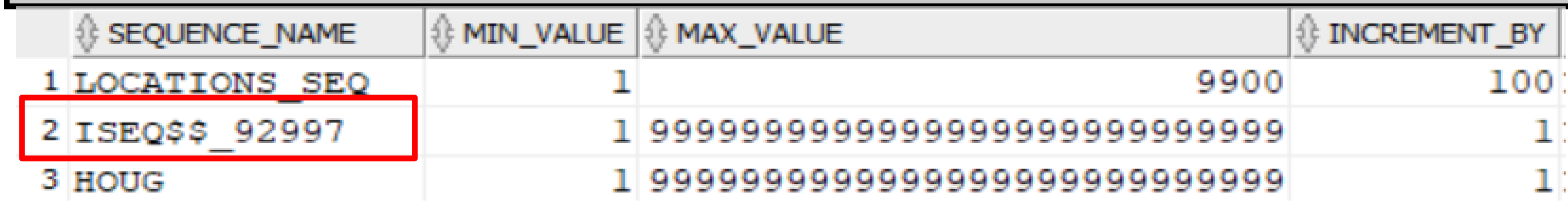
```
DROP TABLE identity_test_tab PURGE;
CREATE TABLE identity_test_tab (  id    NUMBER GENERATED ALWAYS AS IDENTITY,
  DESCRIPTION VARCHAR2(30));
INSERT INTO identity_test_tab(DESCRIPTION) VALUES ('HOUG');
INSERT INTO identity_test_tab(DESCRIPTION) VALUES ('HOUG2019');
COMMIT;
SELECT * FROM identity_test_tab;
-- But!
INSERT INTO identity_test_tab(id,DESCRIPTION) VALUES (3,'HOUG2020');
```

SQL Error: ORA-32795: cannot insert into a generated always identity column

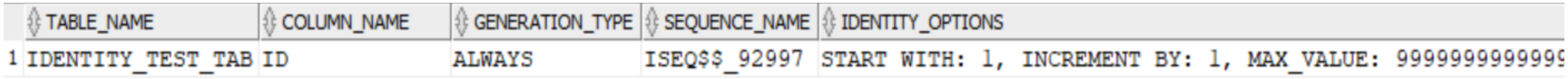
```
SELECT * FROM seq ORDER BY sequence_name DESC;
SELECT * FROM USER_TAB_IDENTITY_COLS;
```



	ID	DESCRIPTION
1	1	HOUG
2	2	HOUG2019



	SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY
1	LOCATIONS_SEQ	1	9900	100
2	ISEQ$$_92997	1	9999999999999999999999999999	1
3	HOUG	1	9999999999999999999999999999	1

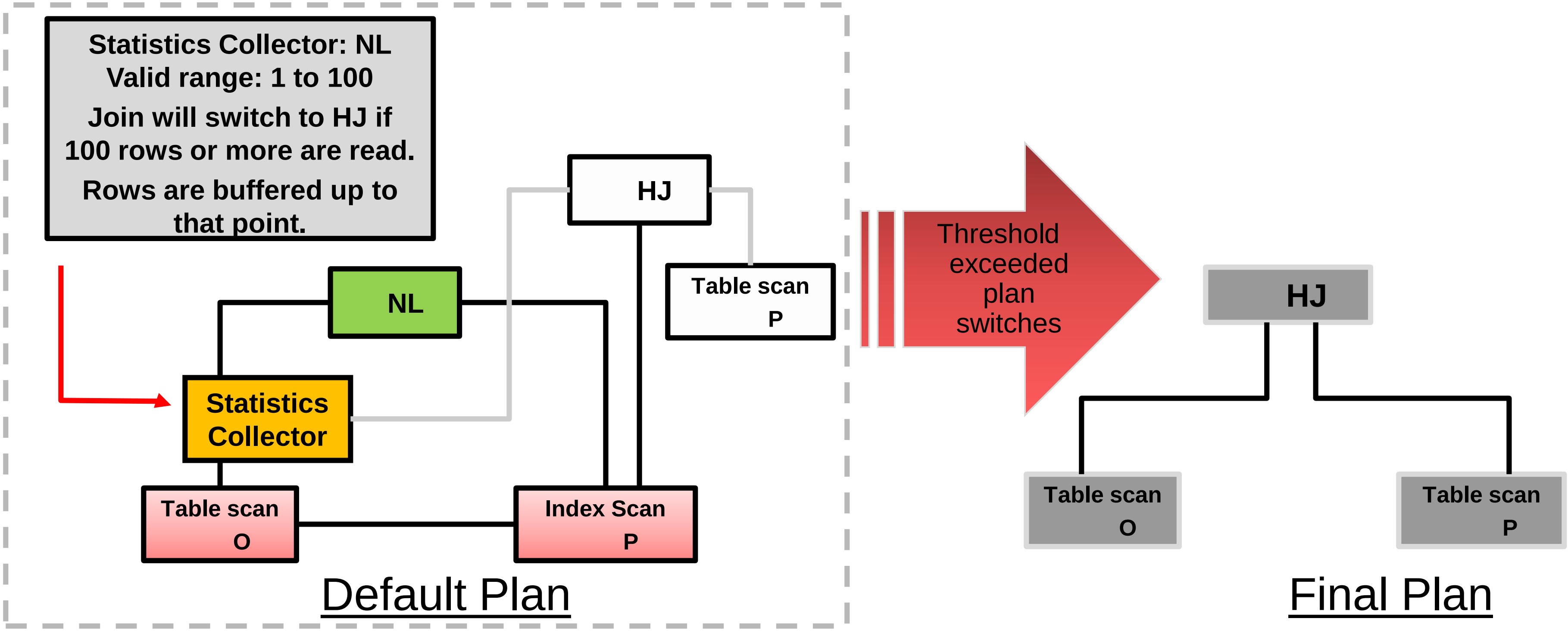


	TABLE_NAME	COLUMN_NAME	GENERATION_TYPE	SEQUENCE_NAME	IDENTITY_OPTIONS
1	IDENTITY_TEST_TAB	ID	ALWAYS	ISEQ$$_92997	START WITH: 1, INCREMENT BY: 1, MAX_VALUE: 999999999999

Adaptive Join Method: Working

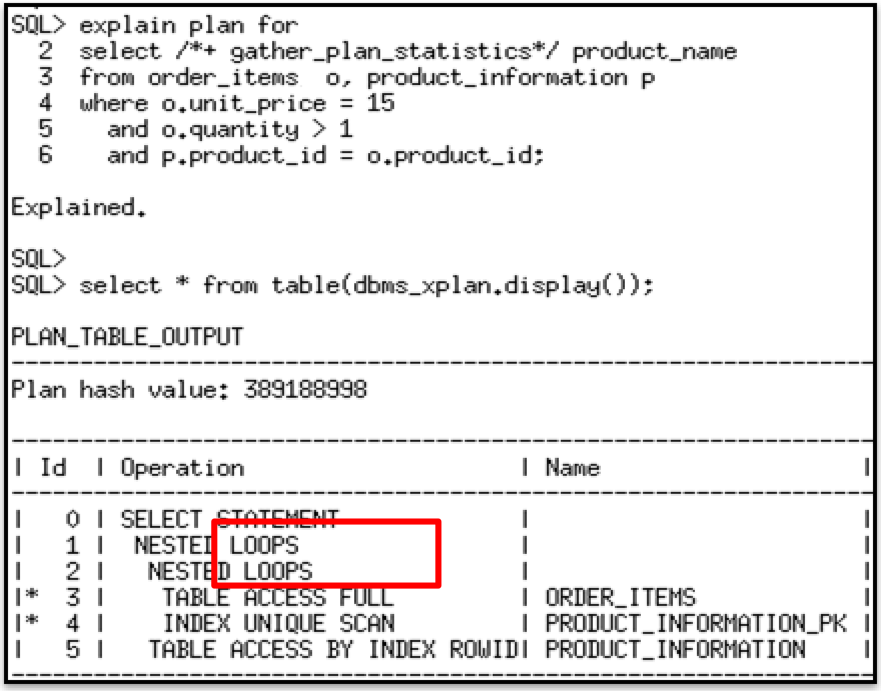
Alternate subplans are pre-computed and stored in the cursor.

- In this case, a nested loops join is replaced by a hash join if the number of rows processed exceeds a valid range.



Displaying the Default Plan

- An explain plan command always shows a default plan.
- The following example shows a nested loops join as the default plan.
- However, there is no statistics collector shown in the plan.



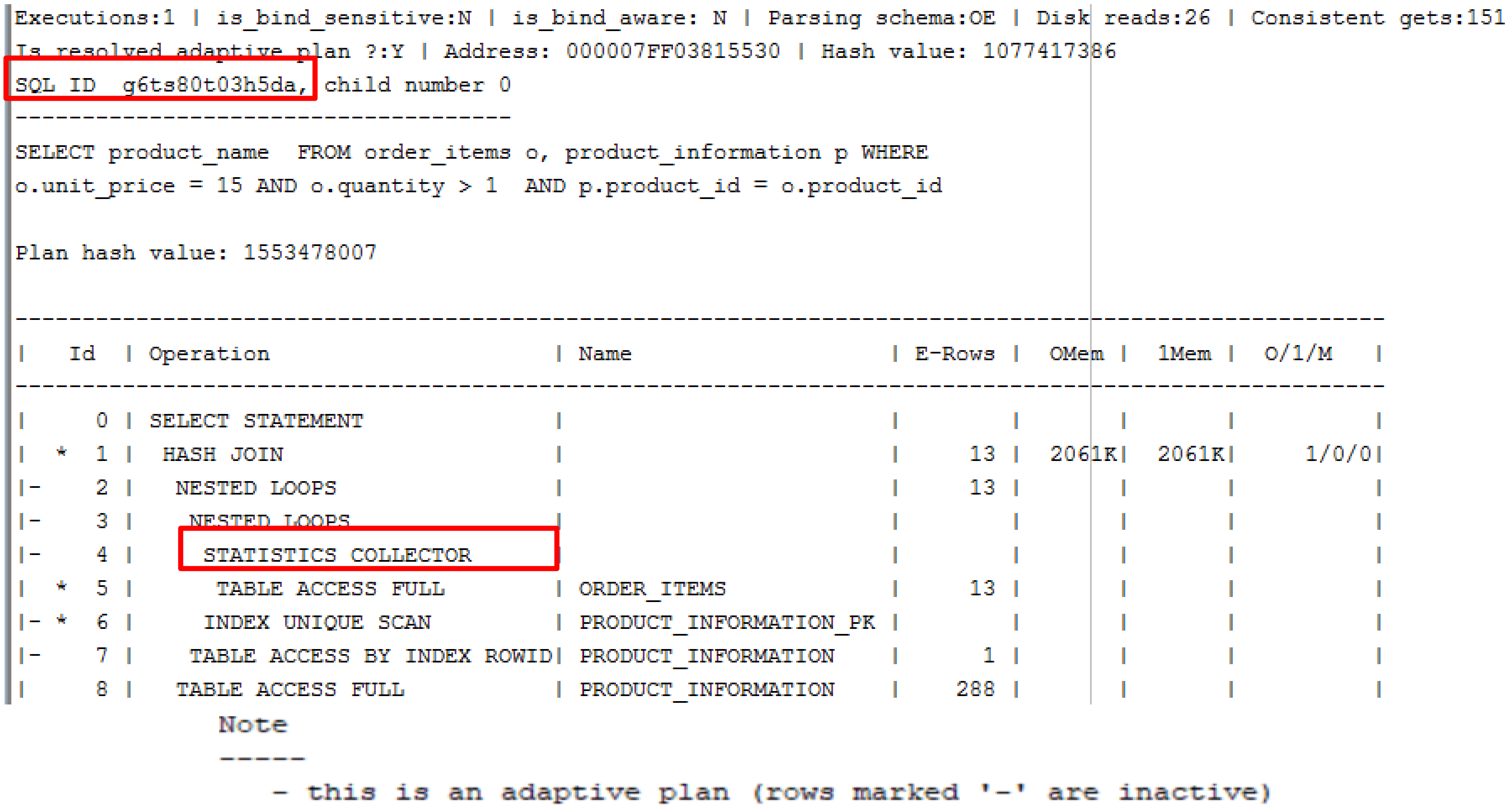
```
SQL> explain plan for
  2  select /*+ gather_plan_statistics*/ product_name
  3  from order_items  o, product_information p
  4  where o.unit_price = 15
  5    and o.quantity > 1
  6    and p.product_id = o.product_id;

Explained.

SQL>
SQL> select * from table(dbms_xplan.display());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------
Plan hash value: 389188998

------------------------------------------------------------------
| Id  | Operation                    | Name                   |
------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                        |
|   1 |  NESTED LOOPS                |                        |
|   2 |   NESTED LOOPS               |                        |
|*  3 |    TABLE ACCESS FULL         | ORDER_ITEMS            |
|*  4 |    INDEX UNIQUE SCAN         | PRODUCT_INFORMATION_PK |
|   5 |   TABLE ACCESS BY INDEX ROWID| PRODUCT_INFORMATION    |
------------------------------------------------------------------
```

```
SELECT product_name
FROM order_items o,
product_information p
WHERE o.unit_price = 15
AND o.quantity > 1
AND p.product_id =
o.product_id
```

Displaying the Full Adaptive Plan

The new adaptive optimization section is shown when the format parameter `+adaptive` is set.

```
exec sqlid('o.unit_price = 15','allstats note adaptive')
```



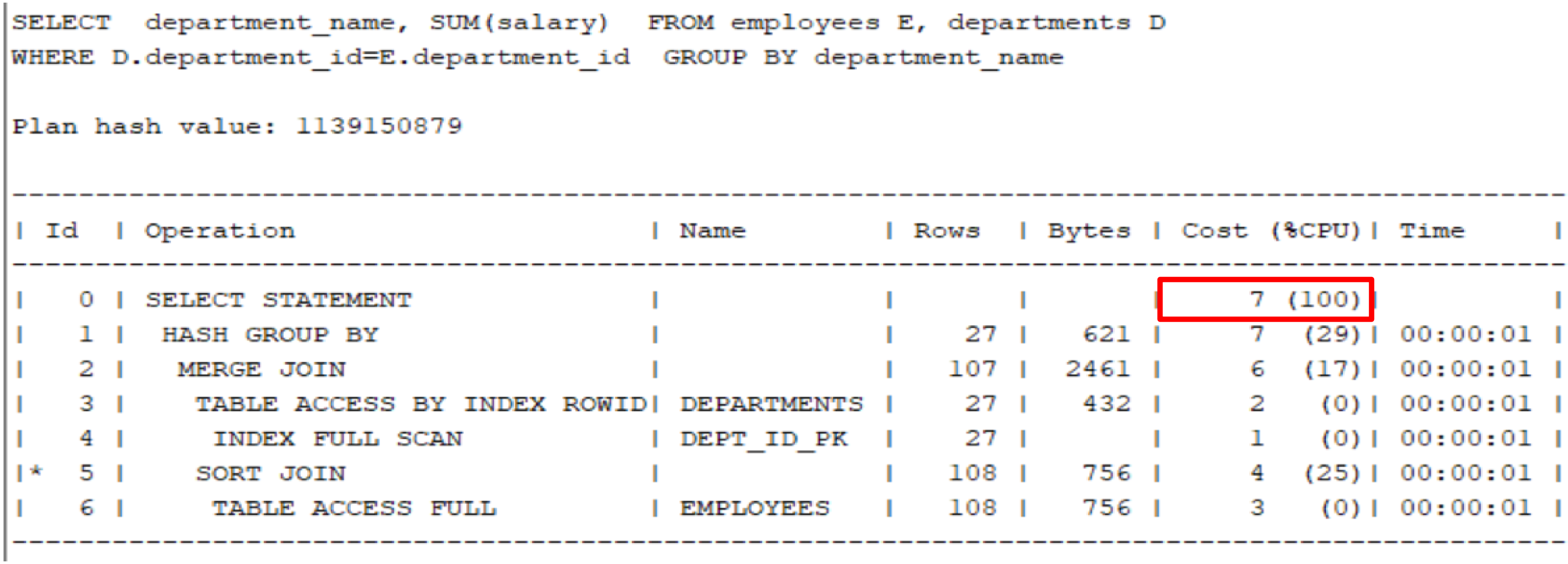
```
Executions:1 | is_bind_sensitive:N | is_bind_aware: N | Parsing schema:OE | Disk reads:26 | Consistent gets:151
Is resolved adaptive plan ?:Y | Address: 000007FF03815530 | Hash value: 1077417386
SQL ID  g6ts80t03h5da, child number 0
-------------------------------------
SELECT product_name  FROM order_items o, product_information p WHERE
o.unit_price = 15 AND o.quantity > 1  AND p.product_id = o.product_id

Plan hash value: 1553478007


----------------------------------------------------------------------------------------------------------
|   Id  | Operation                    | Name                  | E-Rows |  OMem |  1Mem |  O/1/M   |
----------------------------------------------------------------------------------------------------------
|     0 | SELECT STATEMENT             |                       |        |       |       |          |
|  *  1 |  HASH JOIN                   |                       |     13 | 2061K|  2061K|     1/0/0|
|-    2 |   NESTED LOOPS               |                       |     13 |       |       |          |
|-    3 |    NESTED LOOPS              |                       |        |       |       |          |
|-    4 |     STATISTICS COLLECTOR     |                       |        |       |       |          |
|  *  5 |      TABLE ACCESS FULL       | ORDER_ITEMS           |     13 |       |       |          |
|- *  6 |     INDEX UNIQUE SCAN        | PRODUCT_INFORMATION_PK |       |       |       |          |
|-    7 |    TABLE ACCESS BY INDEX ROWID| PRODUCT_INFORMATION  |      1 |       |       |          |
|     8 |   TABLE ACCESS FULL          | PRODUCT_INFORMATION   |    288 |       |       |          |
              Note
              -----
                 - this is an adaptive plan (rows marked '-' are inactive)
```

Adaptive Plans: Parallel Distribution Method

- Parallel execution requires data redistribution to perform operations such as parallel sorts, aggregations, and joins.
- Data distribution is necessary when parallel execution is used.
- The decision on distribution method is based on operation and expected number of rows.
- A new adaptive distribution method is HYBRID-HASH.
  - Statistics collectors are inserted in front of the parallel server process on the left side of the join.
  - If the actual number of rows is less than a threshold, there is a switch from hash distribution to broadcast.

Example
(without PARALLEL hint)

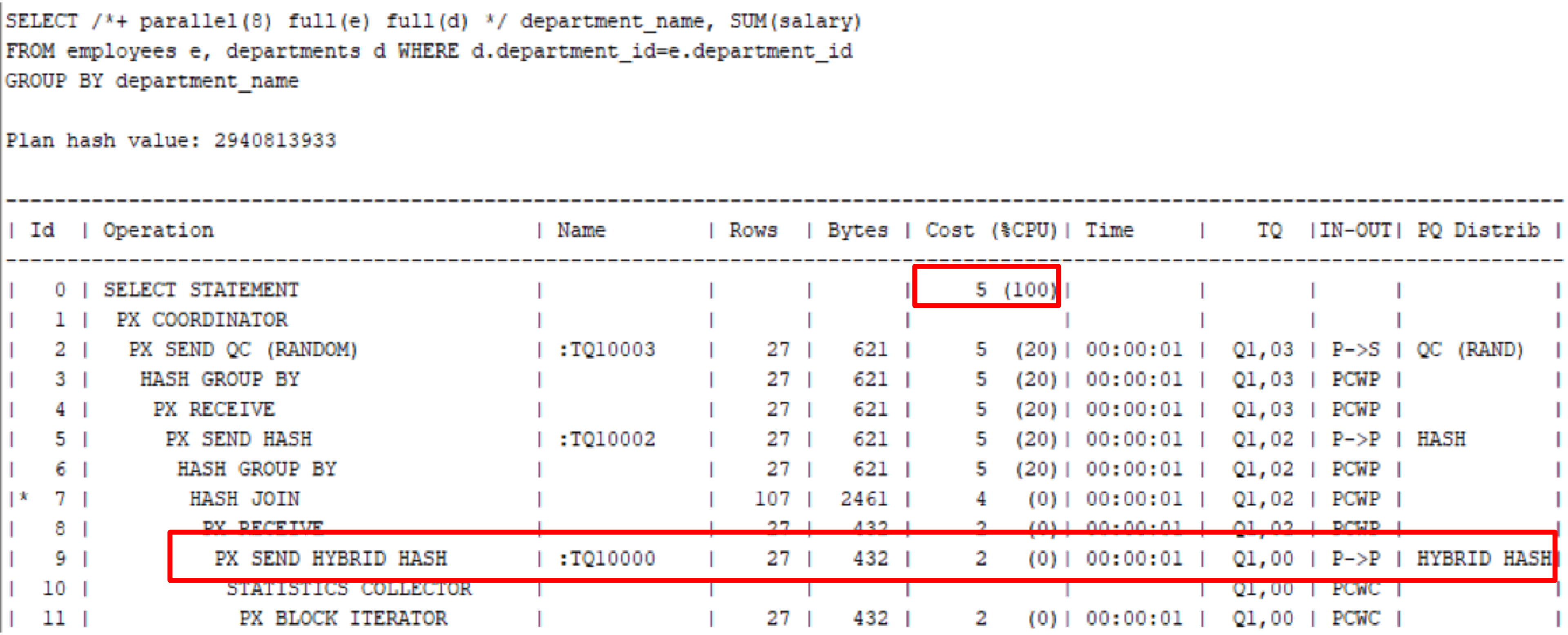
```
SELECT  department_name, SUM(salary)
FROM employees E, departments D WHERE
D.department_id=E.department_id
GROUP BY department_name;
```



```
SELECT  department_name, SUM(salary)  FROM employees E, departments D
WHERE D.department_id=E.department_id  GROUP BY department_name

Plan hash value: 1139150879

----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost  (%CPU)| Time     |
----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |       |       |     7  (100)|          |
|   1 |  HASH GROUP BY                |               |    27 |   621 |     7   (29)| 00:00:01 |
|   2 |   MERGE JOIN                  |               |   107 |  2461 |     6   (17)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID| DEPARTMENTS   |    27 |   432 |     2    (0)| 00:00:01 |
|   4 |     INDEX FULL SCAN           | DEPT_ID_PK    |    27 |       |     1    (0)| 00:00:01 |
|*  5 |    SORT JOIN                  |               |   108 |   756 |     4   (25)| 00:00:01 |
|   6 |     TABLE ACCESS FULL         | EMPLOYEES     |   108 |   756 |     3    (0)| 00:00:01 |
----------------------------------------------------------------------------------------------
```

Example
(with PARALLEL hint)

```
SELECT /*+ parallel(8) full(e) full(d) */ department_name,
SUM(salary)
FROM employees e, departments d WHERE
d.department_id=e.department_id
GROUP BY department_name;
```



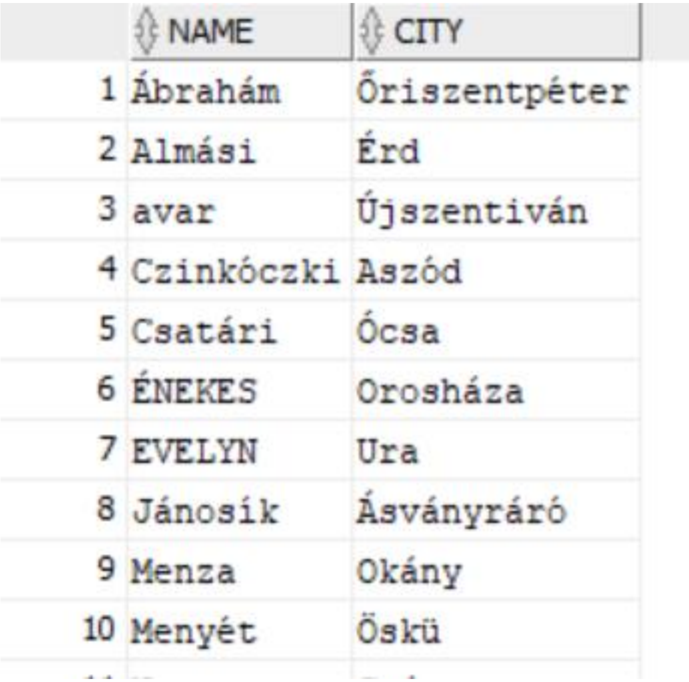
```
SELECT /*+ parallel(8) full(e) full(d) */ department_name, SUM(salary)
FROM employees e, departments d WHERE d.department_id=e.department_id
GROUP BY department_name

Plan hash value: 2940813933

--------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes | Cost (%CPU)| Time     |    TQ  |IN-OUT| PQ Distrib  |
--------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |           |       |       |     5 (100)|          |        |      |             |
|   1 |  PX COORDINATOR               |           |       |       |            |          |        |      |             |
|   2 |   PX SEND QC (RANDOM)         | :TQ10003  |    27 |   621 |     5  (20)| 00:00:01 |  Q1,03 | P->S | QC (RAND)   |
|   3 |    HASH GROUP BY              |           |    27 |   621 |     5  (20)| 00:00:01 |  Q1,03 | PCWP |             |
|   4 |     PX RECEIVE                |           |    27 |   621 |     5  (20)| 00:00:01 |  Q1,03 | PCWP |             |
|   5 |      PX SEND HASH             | :TQ10002  |    27 |   621 |     5  (20)| 00:00:01 |  Q1,02 | P->P | HASH        |
|   6 |       HASH GROUP BY           |           |    27 |   621 |     5  (20)| 00:00:01 |  Q1,02 | PCWP |             |
|*  7 |        HASH JOIN              |           |   107 |  2461 |     4   (0)| 00:00:01 |  Q1,02 | PCWP |             |
|   8 |         PX RECEIVE            |           |    27 |   432 |     2   (0)| 00:00:01 |  Q1,02 | PCWP |             |
|   9 |          PX SEND HYBRID HASH  | :TQ10000  |    27 |   432 |     2   (0)| 00:00:01 |  Q1,00 | P->P | HYBRID HASH |
|  10 |           STATISTICS COLLECTOR |          |       |       |            |          |  Q1,00 | PCWC |             |
|  11 |            PX BLOCK ITERATOR  |           |    27 |   432 |     2   (0)| 00:00:01 |  Q1,00 | PCWC |             |
```

The COLLATE operator

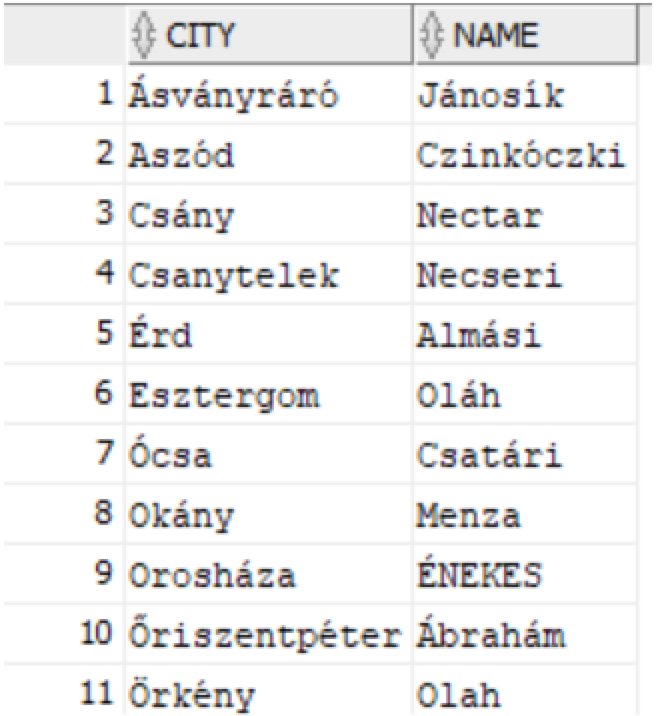
- The COLLATE operator determines the collation for an expression.
- This operator enables you to override the collation that the database would have derived for the expression using standard collation derivation rules.
- You can apply this operator to expressions of type VARCHAR2, CHAR, LONG, NVARCHAR, or NCHAR.

```
SELECT NAME,city
FROM xhun
ORDER BY NAME
COLLATE xhungarian_ai;
```

```
SELECT city,name FROM xhun
ORDER BY city
COLLATE XHungarian_ci,
name COLLATE XHungarian_ci;
```



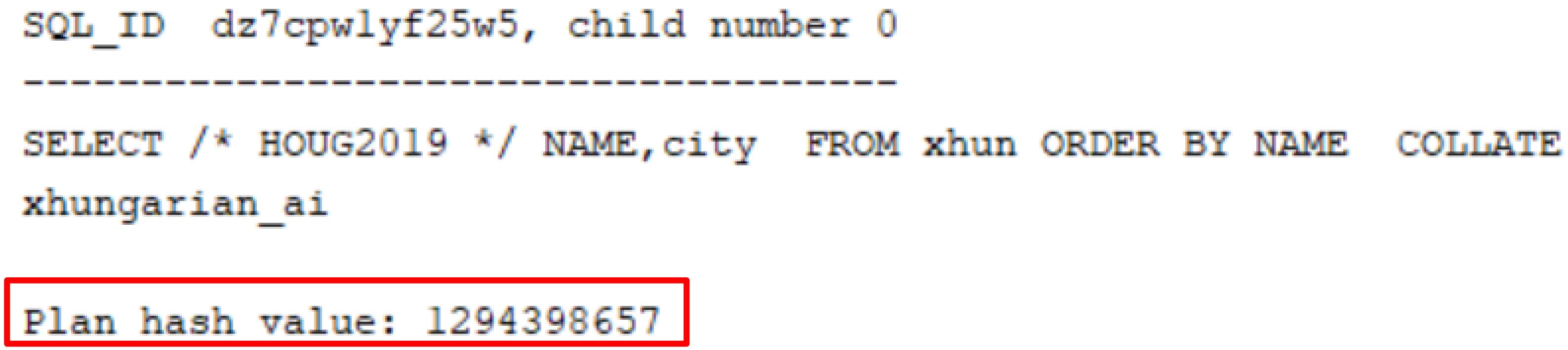
	NAME	CITY
1	Ábrahám	Őriszentpéter
2	Almási	Érd
3	avar	Újszentiván
4	Czinkóczki	Aszód
5	Csatári	Ócsa
6	ÉNEKES	Orosháza
7	EVELYN	Ura
8	Jánosík	Ásványráró
9	Menza	Okány
10	Menyét	Öskü



	CITY	NAME
1	Ásványráró	Jánosík
2	Aszód	Czinkóczki
3	Csány	Nectar
4	Csanytelek	Necseri
5	Érd	Almási
6	Esztergom	Oláh
7	Ócsa	Csatári
8	Okány	Menza
9	Orosháza	ÉNEKES
10	Őriszentpéter	Ábrahám
11	Örkény	Olah

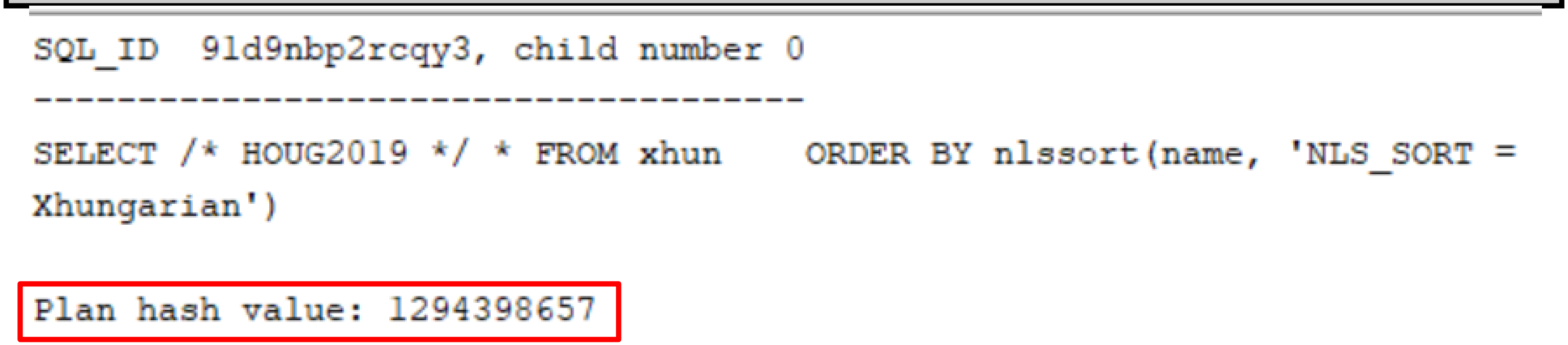
COLLATE versus NLSSORT

```
SELECT /* HOUG2019 */ NAME,city
FROM xhun
ORDER BY NAME COLLATE xhungarian_ai;
```



```
SQL_ID  dz7cpwlyf25w5, child number 0
-------------------------------------
SELECT /* HOUG2019 */ NAME,city  FROM xhun ORDER BY NAME  COLLATE
xhungarian_ai

Plan hash value: 1294398657
```

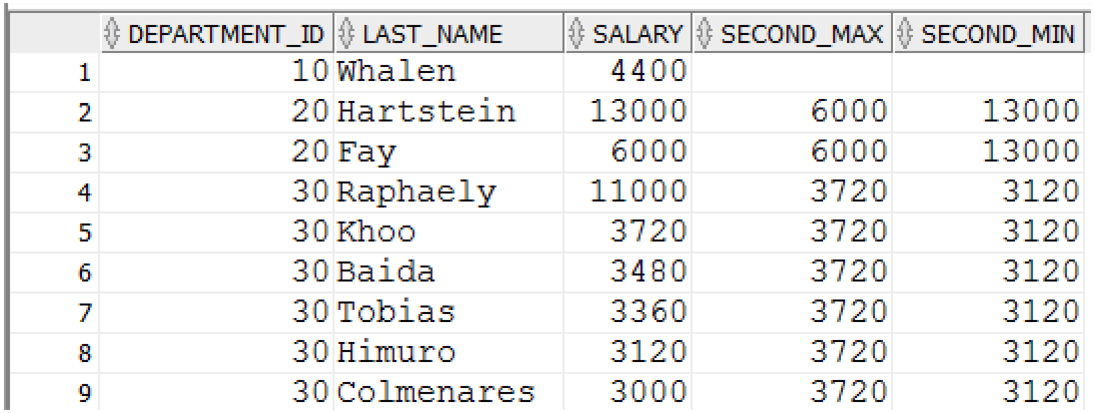
```
SELECT /* HOUG2019 */ * FROM xhun
ORDER BY nlssort(name, 'NLS_SORT = Xhungarian');
EXEC SQLID('/* HOUG2019 */','all')
```



```
SQL_ID  91d9nbp2rcqy3, child number 0
-------------------------------------
SELECT /* HOUG2019 */ * FROM xhun    ORDER BY nlssort(name, 'NLS_SORT =
Xhungarian')

Plan hash value: 1294398657
```

The NTH_VALUE Function

```
SELECT department_id, last_name, salary,
NTH_VALUE(salary,2)
OVER( PARTITION BY department_id ORDER BY salary DESC
ROWS BETWEEN
UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) Second_max,
NTH_VALUE(salary,2)  FROM LAST
OVER( PARTITION BY department_id ORDER BY salary DESC
ROWS BETWEEN
UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) Second_min
FROM employees;
```

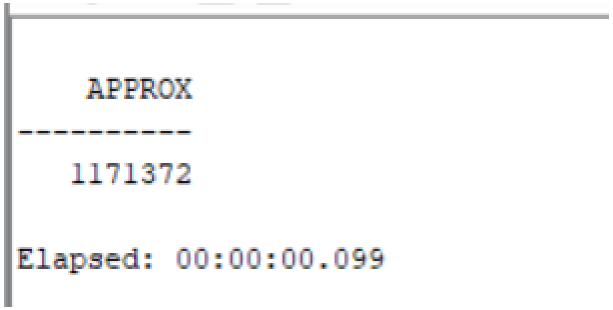


	DEPARTMENT_ID	LAST_NAME	SALARY	SECOND_MAX	SECOND_MIN
1	10	Whalen	4400		
2	20	Hartstein	13000	6000	13000
3	20	Fay	6000	6000	13000
4	30	Raphaely	11000	3720	3120
5	30	Khoo	3720	3720	3120
6	30	Baida	3480	3720	3120
7	30	Tobias	3360	3720	3120
8	30	Himuro	3120	3720	3120
9	30	Colmenares	3000	3720	3120

The APPROX functions in Oracle 12c R2

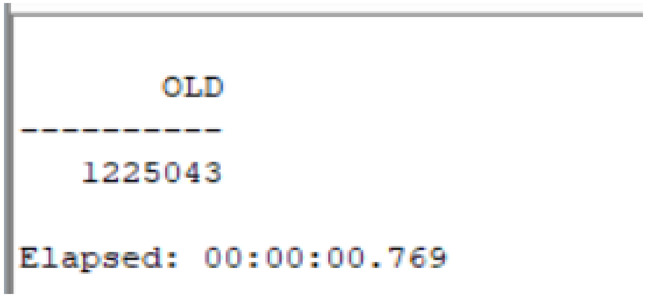
APPROX_COUNT_DISTINCT returns the approximate number of rows that contain a distinct value for *expr*.

```
SELECT APPROX_COUNT_DISTINCT(empno) approx
FROM big_emp;
```



```
    APPROX
----------
   1171372

Elapsed: 00:00:00.099
```

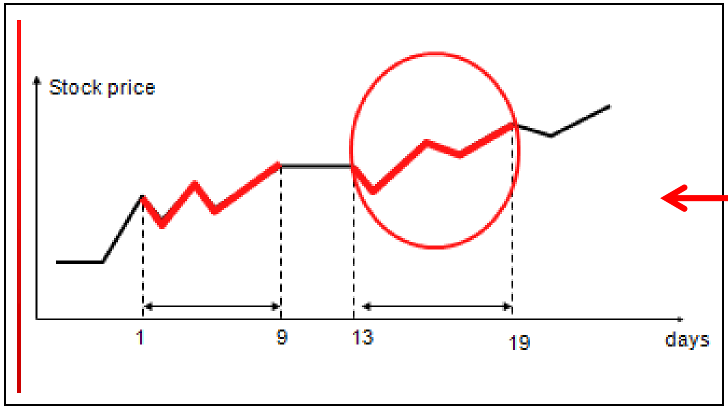
```
SELECT COUNT(DISTINCT empno) old
FROM big_emp;
```



```
       OLD
----------
   1225043

Elapsed: 00:00:00.769
```

Benefits of Pattern Matching

- Pattern matching identifies price patterns, such as V-shapes and W-shapes in stock charts, along with performing many types of calculations.
- The ability to recognize patterns found across multiple rows is essential for many kinds of work:
  - In security applications to detect unusual behavior
  - In financial applications to seek patterns of pricing, trading volume, and other behavior

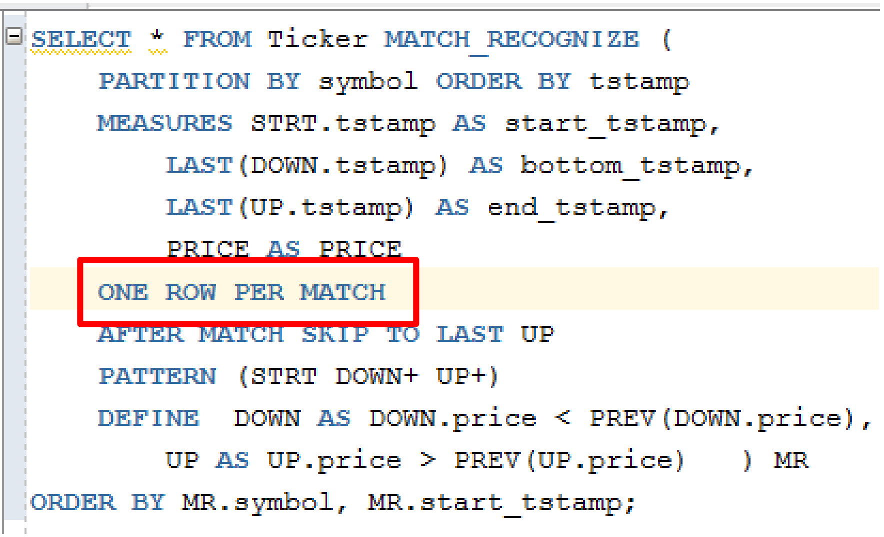


W-shaped patterns in a stock chart

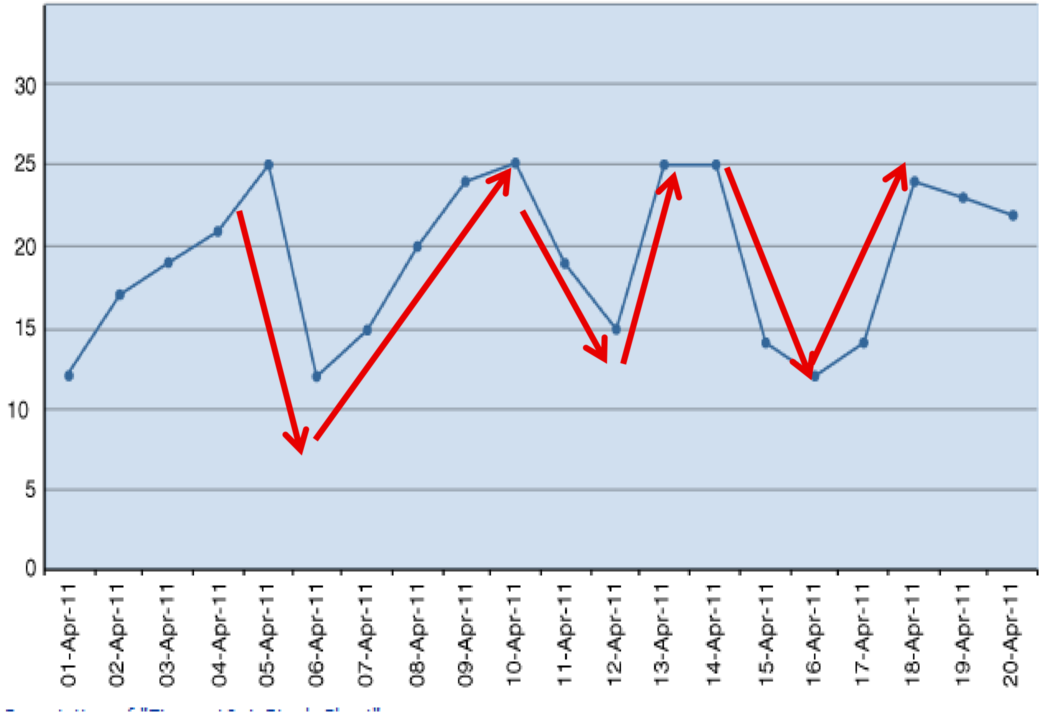
Keywords in Pattern Matching

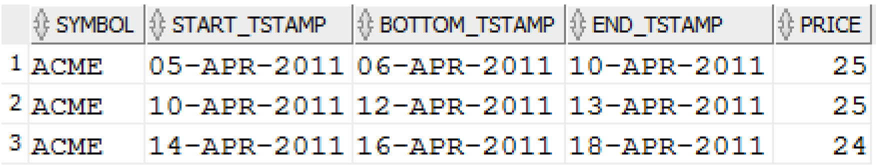
- `PARTITION BY`: Logically divides rows into groups
- `[ONE ROW | ALL ROWS] PER MATCH`: For each row in the match, displays one output row or all output rows
- `MEASURES`: Defines calculations for export from the pattern matching
- `PATTERN`: Defines the row pattern that will be matched
- `DEFINE`: Defines primary pattern variables
- `AFTER MATCH SKIP`: Restarts the matching process after a match is found
- `MATCH_NUMBER`: Finds which rows are members of which match
- `CLASSIFIER`: Finds which pattern variable applies to which rows

Pattern Matching:
Example for ONE ROW PER MATCH



```
SELECT * FROM Ticker MATCH_RECOGNIZE (
    PARTITION BY symbol ORDER BY tstamp
    MEASURES STRT.tstamp AS start_tstamp,
        LAST(DOWN.tstamp) AS bottom_tstamp,
        LAST(UP.tstamp) AS end_tstamp,
        PRICE AS PRICE
    ONE ROW PER MATCH
    AFTER MATCH SKIP TO LAST UP
    PATTERN (STRT DOWN+ UP+)
    DEFINE  DOWN AS DOWN.price < PREV(DOWN.price),
        UP AS UP.price > PREV(UP.price)    ) MR
ORDER BY MR.symbol, MR.start_tstamp;
```

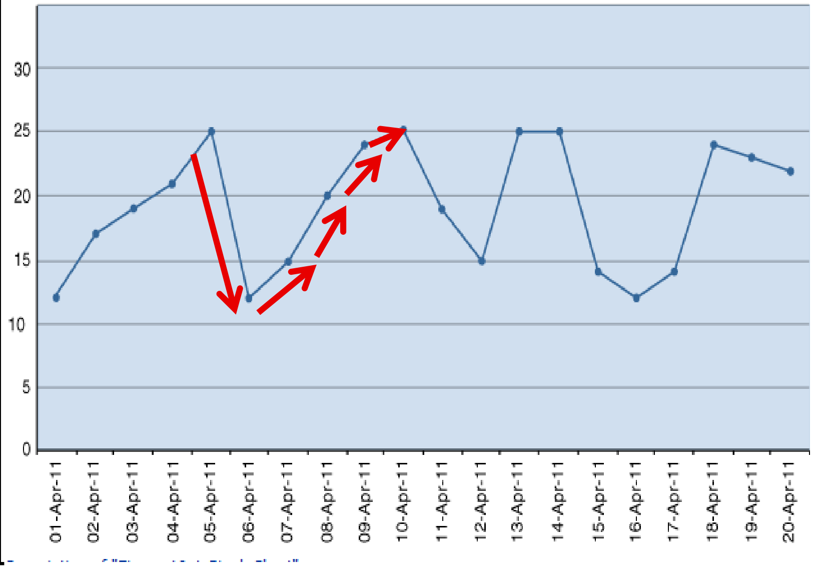


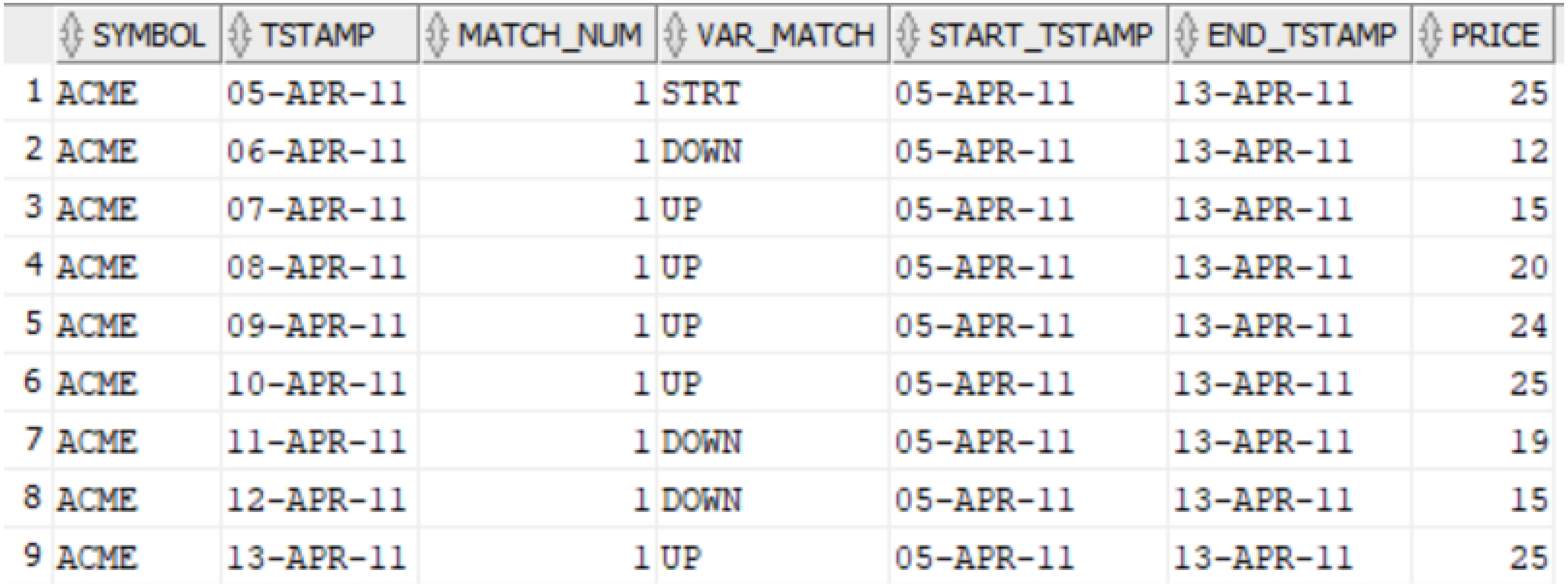


	SYMBOL	START_TSTAMP	BOTTOM_TSTAMP	END_TSTAMP	PRICE
1	ACME	05-APR-2011	06-APR-2011	10-APR-2011	25
2	ACME	10-APR-2011	12-APR-2011	13-APR-2011	25
3	ACME	14-APR-2011	16-APR-2011	18-APR-2011	24

Example for ALL ROWS PER MATCH

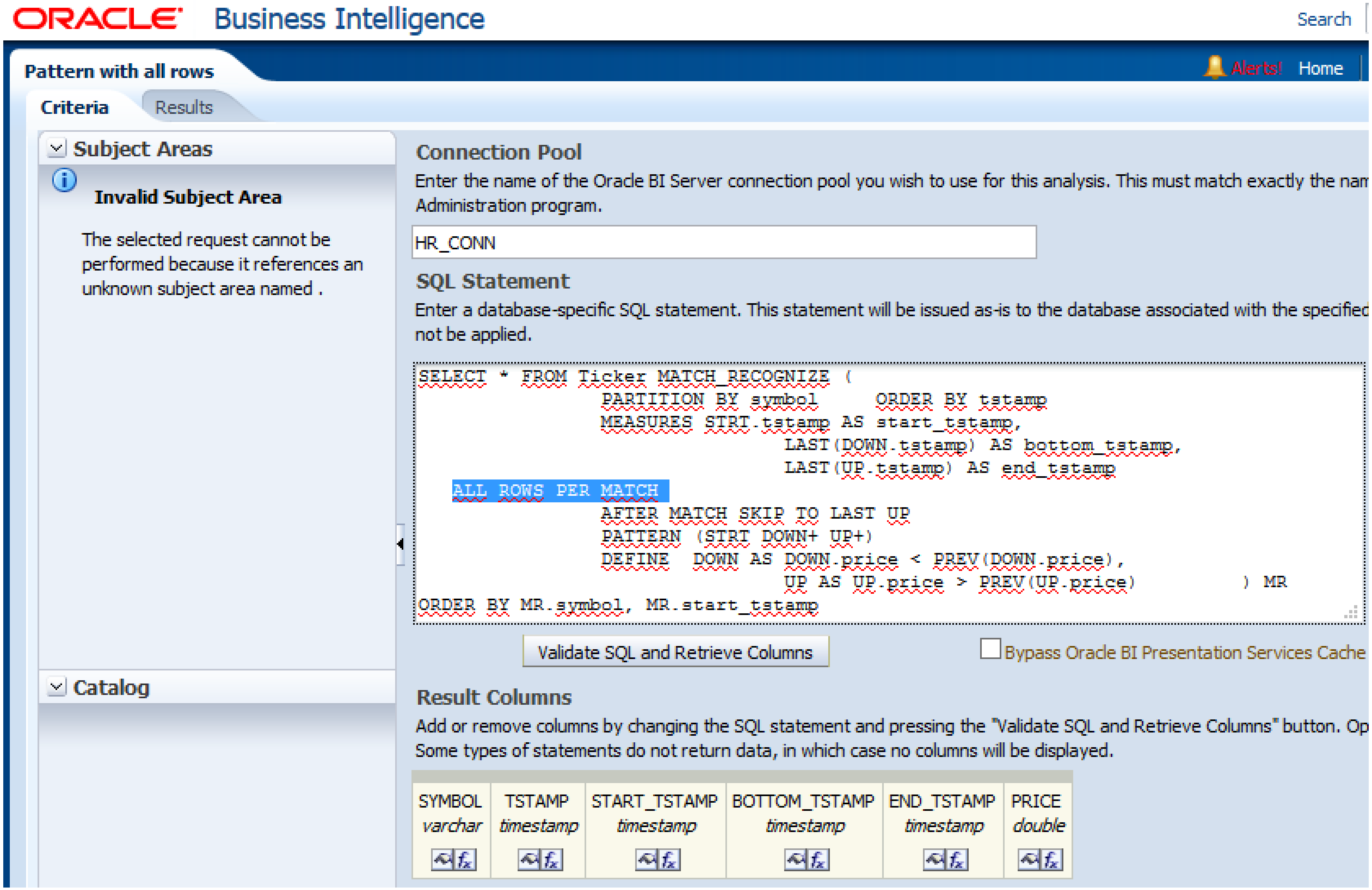
```
SELECT *
FROM ticker MATCH_RECOGNIZE ( PARTITION BY symbol
    ORDER BY tstamp
    MEASURES  strt.tstamp AS start_tstamp,
            CLASSIFIER()  AS  var_match,
            LAST(DOWN.tstamp) AS bottom_tstamp,
            LAST(UP.tstamp) AS end_tstamp
    ALL ROWS PER MATCH
    AFTER MATCH SKIP TO LAST UP
    PATTERN (STRT DOWN+ UP+)
    DEFINE
    down AS down.price < prev(down.price),
    UP AS UP.price > PREV(UP.price) ) mr
ORDER BY MR.symbol, MR.start_tstamp;
```





	SYMBOL	TSTAMP	MATCH_NUM	VAR_MATCH	START_TSTAMP	END_TSTAMP	PRICE
1	ACME	05-APR-11	1	STRT	05-APR-11	13-APR-11	25
2	ACME	06-APR-11	1	DOWN	05-APR-11	13-APR-11	12
3	ACME	07-APR-11	1	UP	05-APR-11	13-APR-11	15
4	ACME	08-APR-11	1	UP	05-APR-11	13-APR-11	20
5	ACME	09-APR-11	1	UP	05-APR-11	13-APR-11	24
6	ACME	10-APR-11	1	UP	05-APR-11	13-APR-11	25
7	ACME	11-APR-11	1	DOWN	05-APR-11	13-APR-11	19
8	ACME	12-APR-11	1	DOWN	05-APR-11	13-APR-11	15
9	ACME	13-APR-11	1	UP	05-APR-11	13-APR-11	25

Example for ALL ROWS PER MATCH in Oracle BI EE (Direct Access)



Invalid Subject Area

The selected request cannot be performed because it references an unknown subject area named .

Connection Pool

Enter the name of the Oracle BI Server connection pool you wish to use for this analysis. This must match exactly the nam
Administration program.

HR_CONN

SQL Statement

Enter a database-specific SQL statement. This statement will be issued as-is to the database associated with the specified
not be applied.

```
SELECT * FROM Ticker MATCH_RECOGNIZE (
                PARTITION BY symbol      ORDER BY tstamp
                MEASURES STRT.tstamp AS start_tstamp,
                                LAST(DOWN.tstamp) AS bottom_tstamp,
                                LAST(UP.tstamp) AS end_tstamp
   ALL ROWS PER MATCH
                AFTER MATCH SKIP TO LAST UP
                PATTERN (STRT DOWN+ UP+)
                DEFINE  DOWN AS DOWN.price < PREV(DOWN.price),
                                UP AS UP.price > PREV(UP.price)          ) MR
ORDER BY MR.symbol, MR.start_tstamp
```

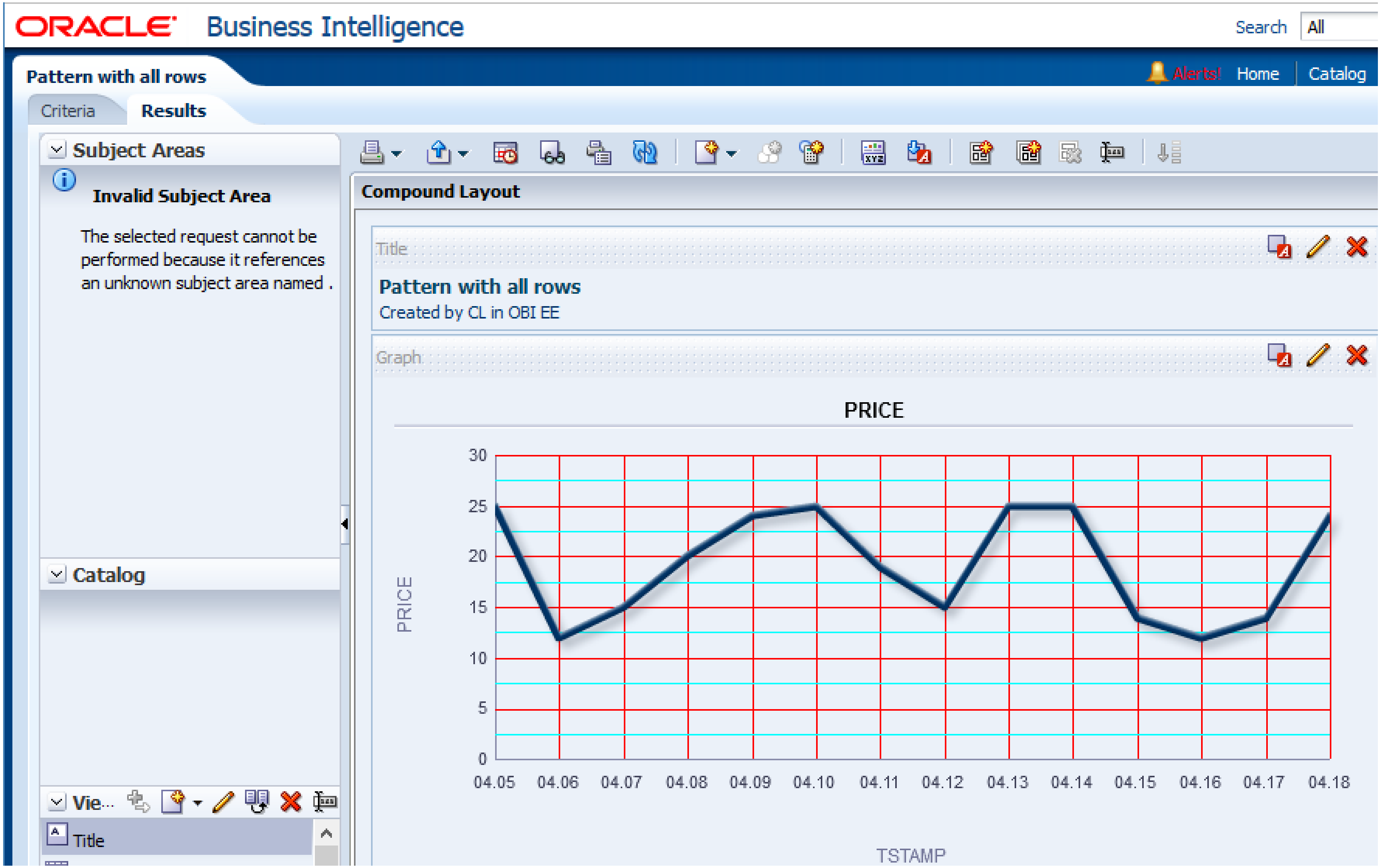
Validate SQL and Retrieve Columns
□ Bypass Oracle BI Presentation Services Cache

Result Columns

Add or remove columns by changing the SQL statement and pressing the "Validate SQL and Retrieve Columns" button. Op
Some types of statements do not return data, in which case no columns will be displayed.

SYMBOL	TSTAMP	START_TSTAMP	BOTTOM_TSTAMP	END_TSTAMP	PRICE
varchar	*timestamp*	*timestamp*	*timestamp*	*timestamp*	*double*

Example for ALL ROWS PER MATCH
in Oracle BI EE (Graph with Line type)



Invalid Subject Area

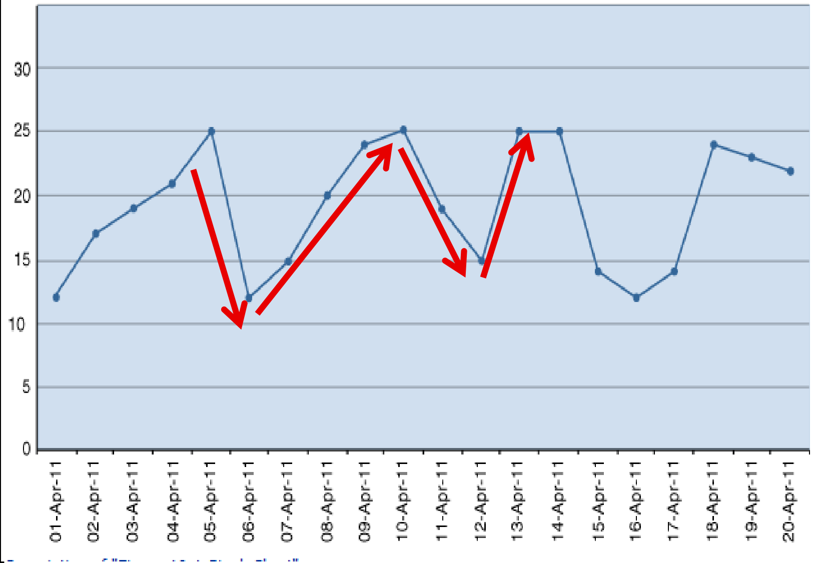
The selected request cannot be performed because it references an unknown subject area named .

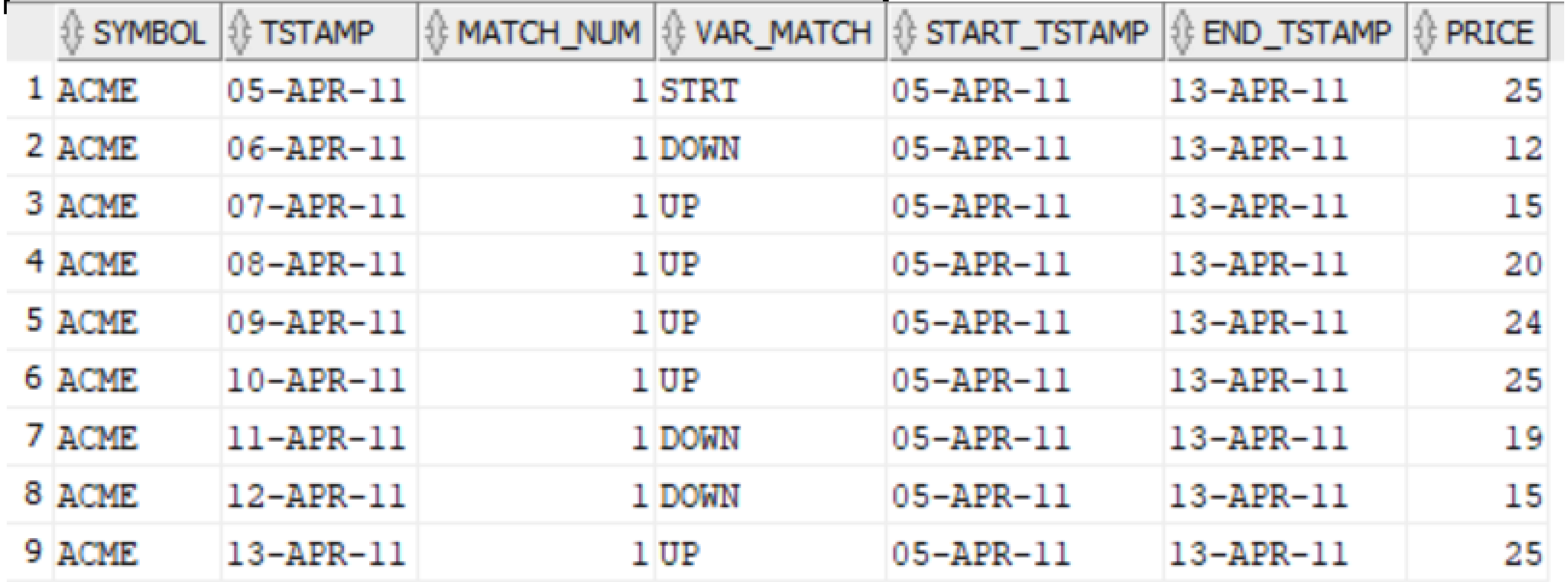
Pattern with all rows

Created by CL in OBI EE

Example for W shape

```
SELECT *
FROM Ticker MATCH_RECOGNIZE ( PARTITION BY
          symbol
          ORDER BY tstamp
          MEASURES
          MATCH_NUMBER() AS match_num,
          CLASSIFIER()  AS  var_match,
          STRT.tstamp AS start_tstamp,
          FINAL LAST(UP.tstamp) AS end_tstamp
          all rows PER MATCH
          AFTER MATCH SKIP TO LAST UP
          PATTERN (STRT DOWN+ UP+ DOWN+ UP+)
          DEFINE
          DOWN AS DOWN.price < PREV(DOWN.price),
          UP AS UP.price > PREV(UP.price) ) MR
ORDER BY mr.symbol, mr.match_num, mr.tstamp;
```





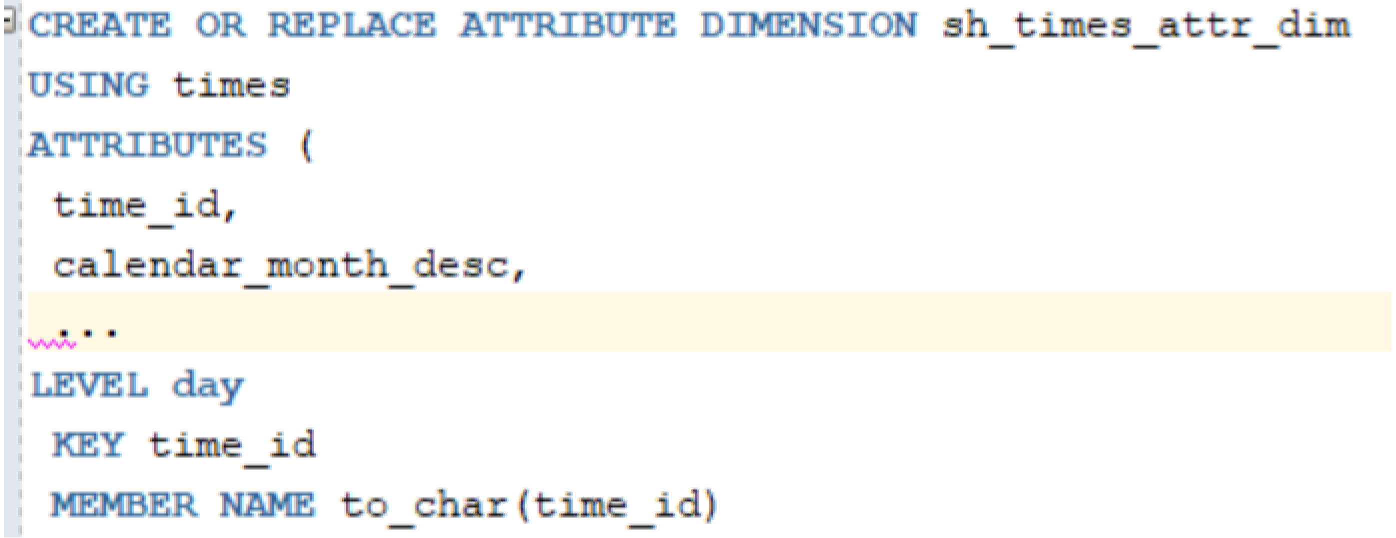
	SYMBOL	TSTAMP	MATCH_NUM	VAR_MATCH	START_TSTAMP	END_TSTAMP	PRICE
1	ACME	05-APR-11	1	STRT	05-APR-11	13-APR-11	25
2	ACME	06-APR-11	1	DOWN	05-APR-11	13-APR-11	12
3	ACME	07-APR-11	1	UP	05-APR-11	13-APR-11	15
4	ACME	08-APR-11	1	UP	05-APR-11	13-APR-11	20
5	ACME	09-APR-11	1	UP	05-APR-11	13-APR-11	24
6	ACME	10-APR-11	1	UP	05-APR-11	13-APR-11	25
7	ACME	11-APR-11	1	DOWN	05-APR-11	13-APR-11	19
8	ACME	12-APR-11	1	DOWN	05-APR-11	13-APR-11	15
9	ACME	13-APR-11	1	UP	05-APR-11	13-APR-11	25

What is the ANALYTIC VIEW?

- An analytic view:
  specifies the source of its fact data and defines measures that describe calculations or other analytic operations to perform on the data.
- Why to use it?
  It work with Oracle SQL engine! (No OLAP, no Essbase)
- An analytic view also specifies the attribute dimensions and hierarchies that define the rows of the analytic view.
- Use the CREATE ANALYTIC VIEW statement to create an analytic view.
- To create an analytic view in your own schema, you must have the CREATE ANALYTIC VIEW system privilege.

How to create analytic view?
1.Create CREATE ATTRIBUTE DIMENSION

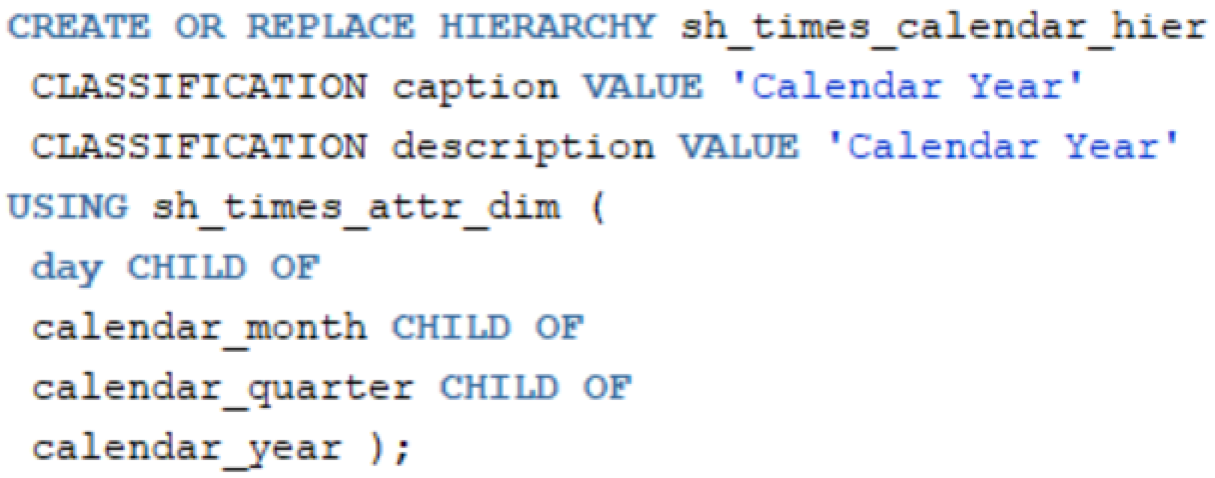
- Use the CREATE ATTRIBUTE DIMENSION statement to create an attribute dimension.
- An attribute dimension specifies dimension members for one or more analytic view hierarchies.
- It specifies the data source it is using and the members it includes.
- It specifies levels for its members and determines attribute relationships between levels.



```
CREATE OR REPLACE ATTRIBUTE DIMENSION sh_times_attr_dim
USING times
ATTRIBUTES (
 time_id,
 calendar_month_desc,
 ...
LEVEL day
 KEY time_id
 MEMBER NAME to_char(time_id)
```

How to create analytic view?
2.Create CREATE HIERARHY

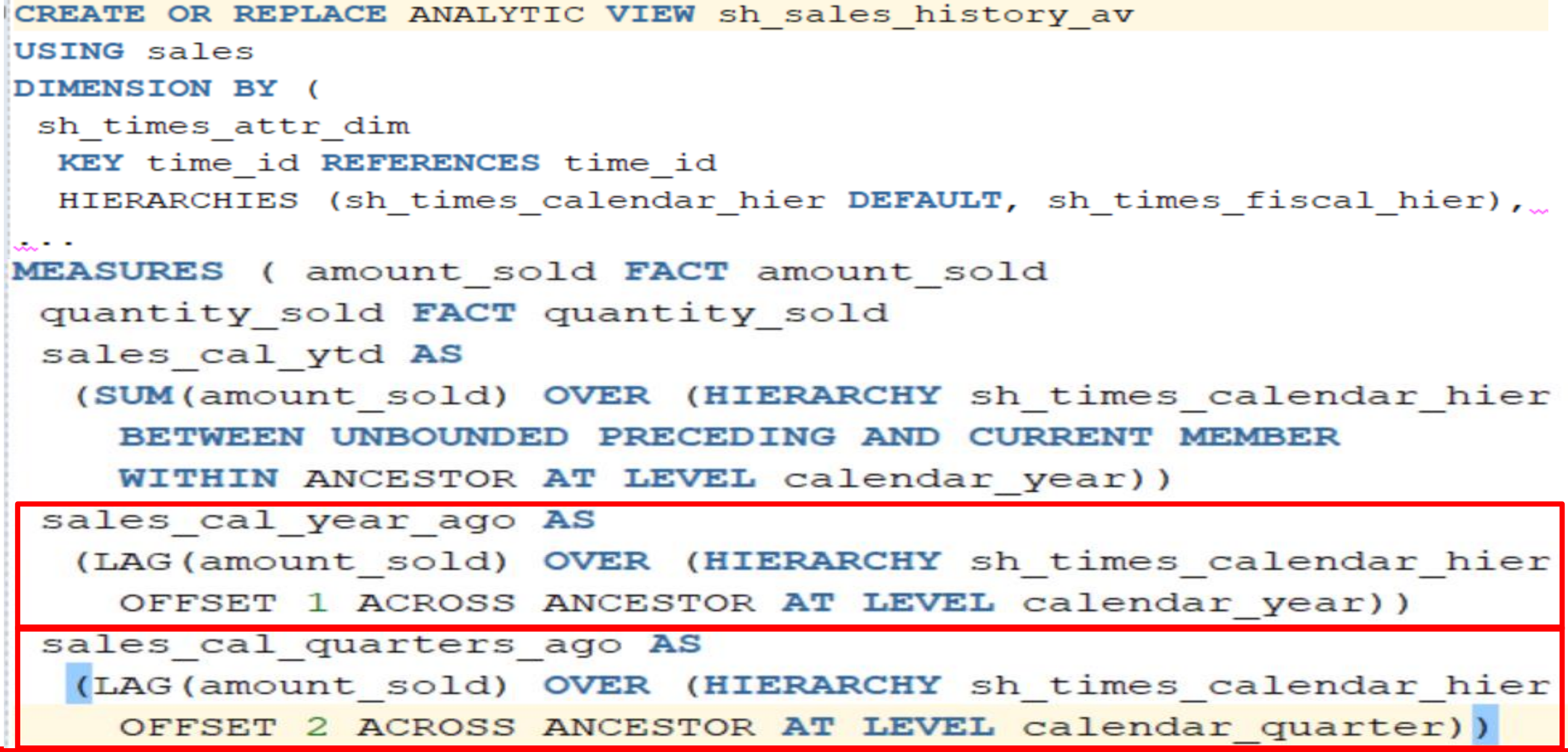
- A hierarchy specifies the hierarchical relationships among the levels of an attribute dimension.
- Use the CREATE HIERARCHY statement to create a hierarchy.
- To create a hierarchy in your own schema,
- You must have the CREATE HIERARCHY system privilege.



```
CREATE OR REPLACE HIERARCHY sh_times_calendar_hier
 CLASSIFICATION caption VALUE 'Calendar Year'
 CLASSIFICATION description VALUE 'Calendar Year'
USING sh_times_attr_dim (
 day CHILD OF
 calendar_month CHILD OF
 calendar_quarter CHILD OF
 calendar_year );
```

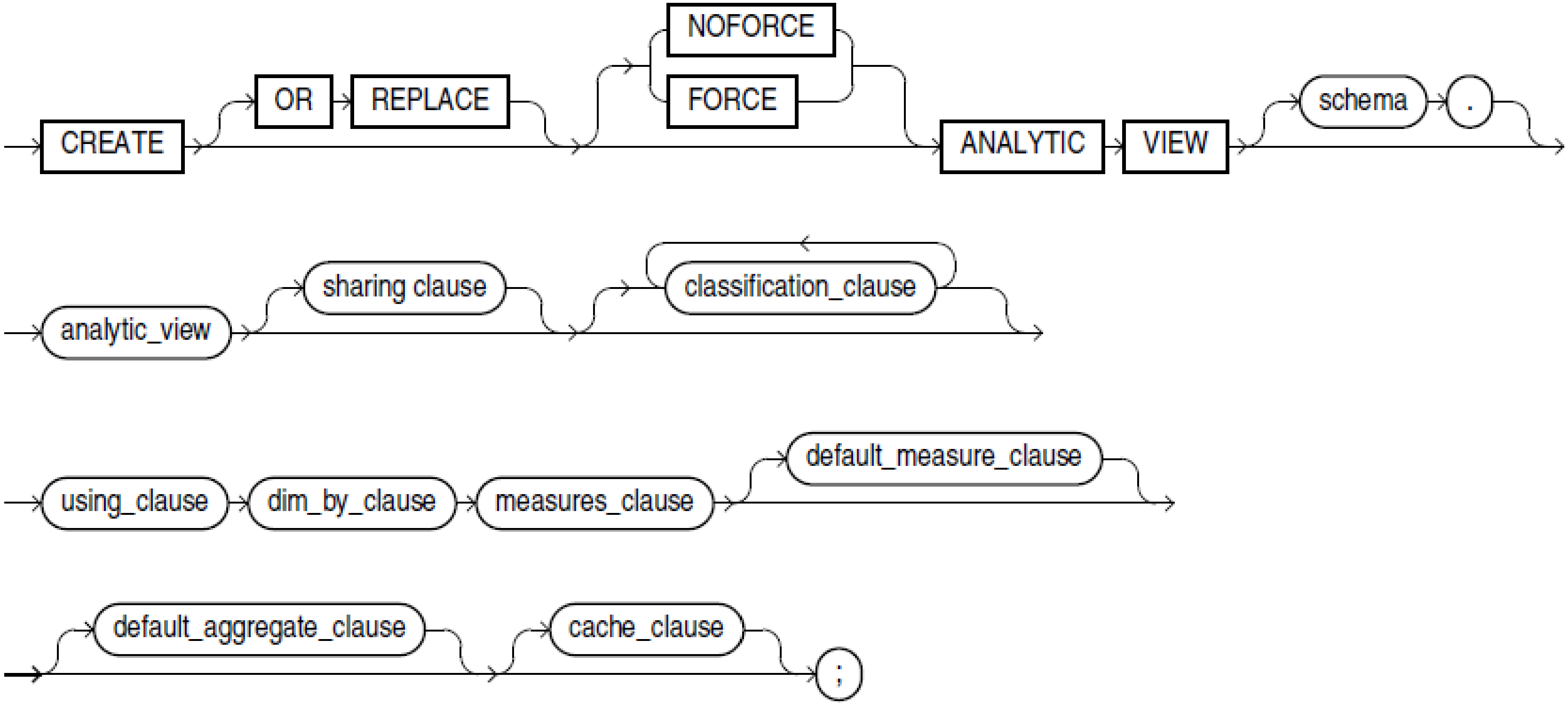
How to create analytic view?
3.Create CREATE ANALYTIC VIEW

- An analytic view specifies the source of its fact data and defines measures that describe calculations or other analytic operations to perform on the data.
- An analytic view also specifies the attribute dimensions and hierarchies that define the rows of the analytic view.
- To create a hierarchy in your own schema, you must have the CREATE HIERARCHY system privilege.Use the CREATE ANALYTIC VIEW statement to create an analytic view.



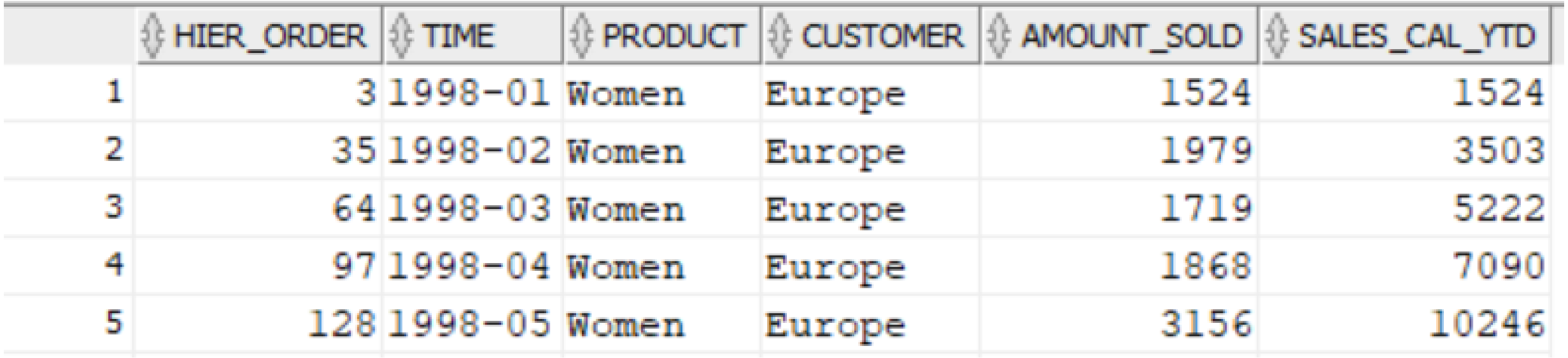
```
CREATE OR REPLACE ANALYTIC VIEW sh_sales_history_av
USING sales
DIMENSION BY (
 sh_times_attr_dim
  KEY time_id REFERENCES time_id
  HIERARCHIES (sh_times_calendar_hier DEFAULT, sh_times_fiscal_hier),
...
MEASURES ( amount_sold FACT amount_sold
 quantity_sold FACT quantity_sold
 sales_cal_ytd AS
  (SUM(amount_sold) OVER (HIERARCHY sh_times_calendar_hier
    BETWEEN UNBOUNDED PRECEDING AND CURRENT MEMBER
    WITHIN ANCESTOR AT LEVEL calendar_year))
 sales_cal_year_ago AS
  (LAG(amount_sold) OVER (HIERARCHY sh_times_calendar_hier
    OFFSET 1 ACROSS ANCESTOR AT LEVEL calendar_year))
 sales_cal_quarters_ago AS
  (LAG(amount_sold) OVER (HIERARCHY sh_times_calendar_hier
    OFFSET 2 ACROSS ANCESTOR AT LEVEL calendar_quarter))
```

Syntax of the Analytic View (abbreviated form)



View Sales Calendar Year to Date, at the Calendar Month level, for Women in Europe:

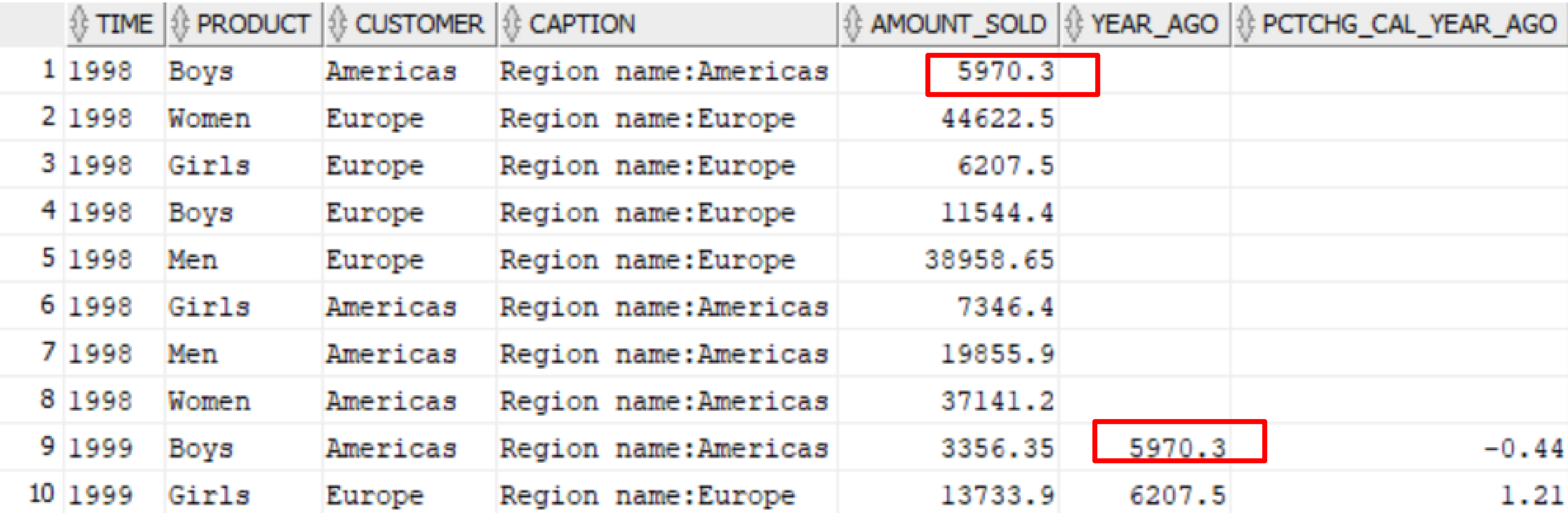
```
SELECT
sh_times_calendar_hier.hier_order,
 sh_times_calendar_hier.member_name AS time,
 sh_products_hier.member_name AS product,
 sh_customers_hier.member_name AS customer,
 amount_sold, sales_cal_ytd
FROM sh_sales_history_av
 HIERARCHIES (  sh_times_calendar_hier,
  sh_products_hier,
 sh_customers_hier )
WHERE
 sh_times_calendar_hier.level_name = 'CALENDAR_MONTH'
 AND sh_products_hier.MEMBER_NAME = 'Women'
 AND sh_customers_hier.member_name = 'Europe'
ORDER BY  sh_times_calendar_hier.HIER_ORDER;
```



	HIER_ORDER	TIME	PRODUCT	CUSTOMER	AMOUNT_SOLD	SALES_CAL_YTD
1	3	1998-01	Women	Europe	1524	1524
2	35	1998-02	Women	Europe	1979	3503
3	64	1998-03	Women	Europe	1719	5222
4	97	1998-04	Women	Europe	1868	7090
5	128	1998-05	Women	Europe	3156	10246

Sales Calendar Year Ago and Sales Percent Change Calendar Year Ago

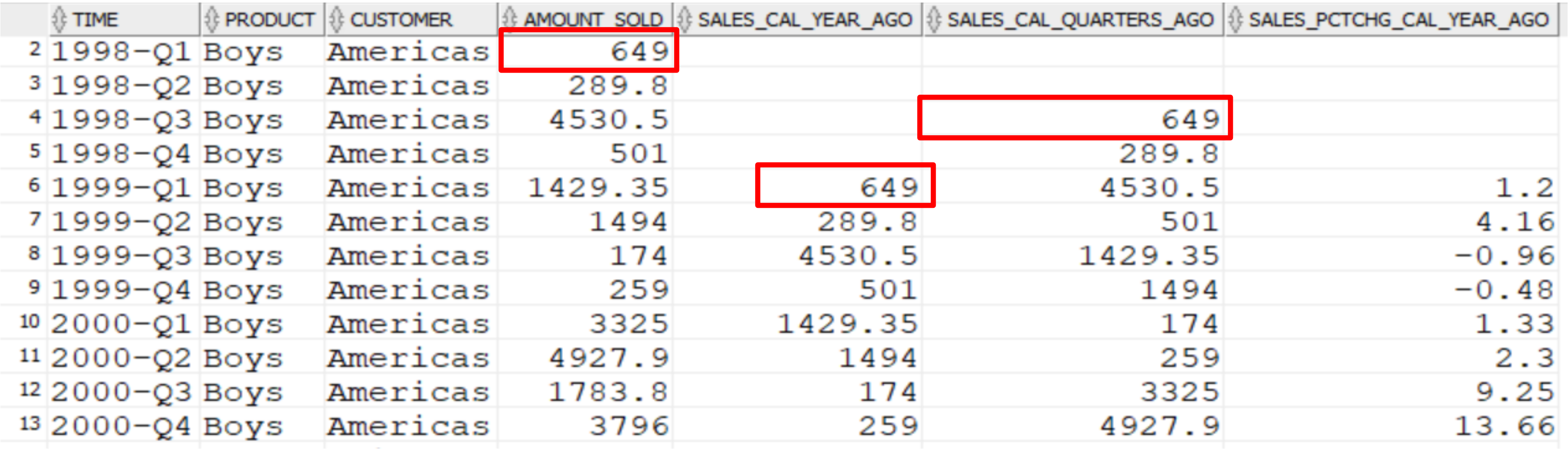
```
SELECT  sh_times_calendar_hier.member_name AS time,
 sh_products_hier.member_name AS product,
 sh_customers_hier.MEMBER_NAME AS customer,
 sh_customers_hier.MEMBER_CAPTION CAPTION,
 amount_sold,  sales_cal_year_ago year_ago,
 ROUND(sales_pctchg_cal_year_ago,2) AS pctchg_cal_year_ago
FROM sh_sales_history_av
 HIERARCHIES (  sh_times_calendar_hier, sh_products_hier,   sh_customers_hier )
WHERE
 sh_times_calendar_hier.level_name = 'CALENDAR_YEAR'
 AND sh_products_hier.level_name = 'CATEGORY'
 AND sh_customers_hier.LEVEL_NAME = 'REGION'
 AND sh_customers_hier.MEMBER_NAME  IN ('Europe','Americas')
ORDER BY  sh_times_calendar_hier.HIER_ORDER;
```



	TIME	PRODUCT	CUSTOMER	CAPTION	AMOUNT_SOLD	YEAR_AGO	PCTCHG_CAL_YEAR_AGO
1	1998	Boys	Americas	Region name:Americas	5970.3		
2	1998	Women	Europe	Region name:Europe	44622.5		
3	1998	Girls	Europe	Region name:Europe	6207.5		
4	1998	Boys	Europe	Region name:Europe	11544.4		
5	1998	Men	Europe	Region name:Europe	38958.65		
6	1998	Girls	Americas	Region name:Americas	7346.4		
7	1998	Men	Americas	Region name:Americas	19855.9		
8	1998	Women	Americas	Region name:Americas	37141.2		
9	1999	Boys	Americas	Region name:Americas	3356.35	5970.3	-0.44
10	1999	Girls	Europe	Region name:Europe	13733.9	6207.5	1.21

Sales Calendar Half Year Ago

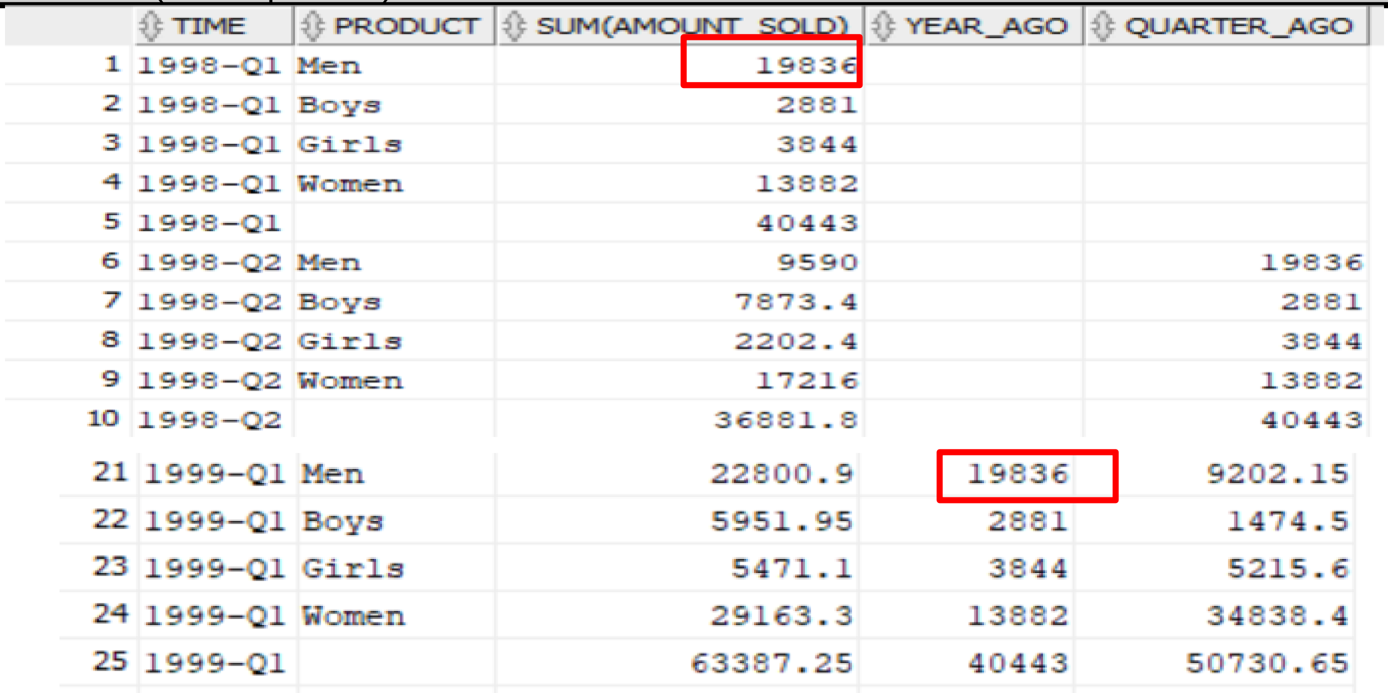
```
SELECT
sh_times_calendar_hier.member_name AS time,
 sh_products_hier.member_name AS product,
 sh_customers_hier.member_name AS customer,
 amount_sold, sales_cal_year_ago, sales_cal_quarters_ago,
 ROUND(sales_pctchg_cal_year_ago,2) AS sales_pctchg_cal_year_ago
FROM sh_sales_history_av
 HIERARCHIES (  sh_times_calendar_hier,   sh_products_hier,  sh_customers_hier )
WHERE
 sh_times_calendar_hier.level_name = 'CALENDAR_QUARTER'
 AND sh_products_hier.level_name = 'CATEGORY'
 AND sh_customers_hier.level_name = 'REGION'
ORDER BY  sh_products_hier.HIER_ORDER,
 sh_customers_hier.HIER_ORDER,  sh_times_calendar_hier.hier_order;
```



	TIME	PRODUCT	CUSTOMER	AMOUNT_SOLD	SALES_CAL_YEAR_AGO	SALES_CAL_QUARTERS_AGO	SALES_PCTCHG_CAL_YEAR_AGO
2	1998-Q1	Boys	Americas	649			
3	1998-Q2	Boys	Americas	289.8			
4	1998-Q3	Boys	Americas	4530.5		649	
5	1998-Q4	Boys	Americas	501		289.8	
6	1999-Q1	Boys	Americas	1429.35	649	4530.5	1.2
7	1999-Q2	Boys	Americas	1494	289.8	501	4.16
8	1999-Q3	Boys	Americas	174	4530.5	1429.35	-0.96
9	1999-Q4	Boys	Americas	259	501	1494	-0.48
10	2000-Q1	Boys	Americas	3325	1429.35	174	1.33
11	2000-Q2	Boys	Americas	4927.9	1494	259	2.3
12	2000-Q3	Boys	Americas	1783.8	174	3325	9.25
13	2000-Q4	Boys	Americas	3796	259	4927.9	13.66

Sales Calendar Quarter Ago with ROLLUP operator

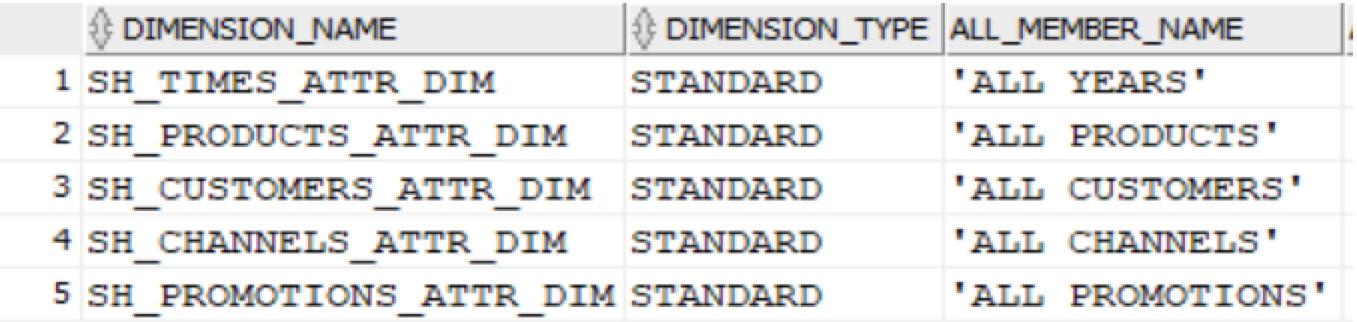
```
SELECT TIME, product,
 SUM(amount_sold),  SUM(sales_cal_year_ago)year_ago,   SUM(sales_cal_quarters_ago) quarter_ago
FROM  (SELECT  sh_times_calendar_hier.member_name AS time,
 sh_products_hier.MEMBER_NAME AS product,
 amount_sold,  sales_cal_year_ago,sales_cal_quarters_ago
FROM sh_sales_history_av_qtr
HIERARCHIES (  sh_times_calendar_hier, sh_products_hier,   sh_customers_hier )
WHERE  sh_times_calendar_hier.level_name = 'CALENDAR_QUARTER'
 AND sh_products_hier.LEVEL_NAME = 'CATEGORY'
AND sh_customers_hier.LEVEL_NAME = 'REGION'
 AND sh_customers_hier.MEMBER_NAME  IN ('Europe','Americas')
ORDER BY   sh_times_calendar_hier.HIER_ORDER)
GROUP BY ROLLUP(TIME, product);
```



	TIME	PRODUCT	SUM(AMOUNT_SOLD)	YEAR_AGO	QUARTER_AGO
1	1998-Q1	Men	19836		
2	1998-Q1	Boys	2881		
3	1998-Q1	Girls	3844		
4	1998-Q1	Women	13882		
5	1998-Q1		40443		
6	1998-Q2	Men	9590		19836
7	1998-Q2	Boys	7873.4		2881
8	1998-Q2	Girls	2202.4		3844
9	1998-Q2	Women	17216		13882
10	1998-Q2		36881.8		40443
21	1999-Q1	Men	22800.9	19836	9202.15
22	1999-Q1	Boys	5951.95	2881	1474.5
23	1999-Q1	Girls	5471.1	3844	5215.6
24	1999-Q1	Women	29163.3	13882	34838.4
25	1999-Q1		63387.25	40443	50730.65

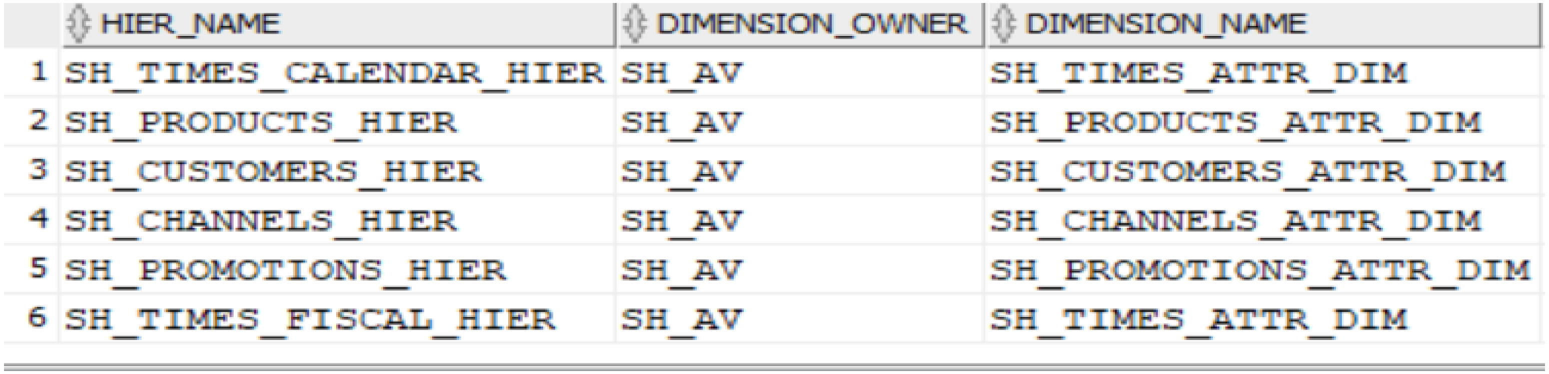
Some useful DD views

```
SELECT * FROM user_attribute_dimensions;
```



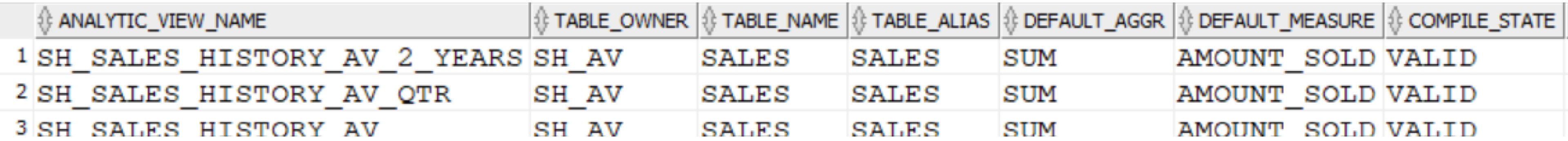
	DIMENSION_NAME	DIMENSION_TYPE	ALL_MEMBER_NAME
1	SH_TIMES_ATTR_DIM	STANDARD	'ALL YEARS'
2	SH_PRODUCTS_ATTR_DIM	STANDARD	'ALL PRODUCTS'
3	SH_CUSTOMERS_ATTR_DIM	STANDARD	'ALL CUSTOMERS'
4	SH_CHANNELS_ATTR_DIM	STANDARD	'ALL CHANNELS'
5	SH_PROMOTIONS_ATTR_DIM	STANDARD	'ALL PROMOTIONS'

```
SELECT * FROM user_hierarchies;
```



	HIER_NAME	DIMENSION_OWNER	DIMENSION_NAME
1	SH_TIMES_CALENDAR_HIER	SH_AV	SH_TIMES_ATTR_DIM
2	SH_PRODUCTS_HIER	SH_AV	SH_PRODUCTS_ATTR_DIM
3	SH_CUSTOMERS_HIER	SH_AV	SH_CUSTOMERS_ATTR_DIM
4	SH_CHANNELS_HIER	SH_AV	SH_CHANNELS_ATTR_DIM
5	SH_PROMOTIONS_HIER	SH_AV	SH_PROMOTIONS_ATTR_DIM
6	SH_TIMES_FISCAL_HIER	SH_AV	SH_TIMES_ATTR_DIM

```
SELECT * FROM user_ANALYTIC_views;
```



	ANALYTIC_VIEW_NAME	TABLE_OWNER	TABLE_NAME	TABLE_ALIAS	DEFAULT_AGGR	DEFAULT_MEASURE	COMPILE_STATE
1	SH_SALES_HISTORY_AV_2_YEARS	SH_AV	SALES	SALES	SUM	AMOUNT_SOLD	VALID
2	SH_SALES_HISTORY_AV_QTR	SH_AV	SALES	SALES	SUM	AMOUNT_SOLD	VALID
3	SH_SALES_HISTORY_AV	SH_AV	SALES	SALES	SUM	AMOUNT_SOLD	VALID

PL/SQL New Features

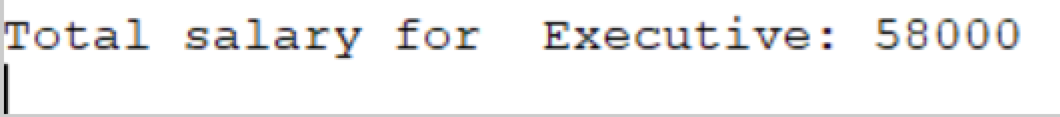
- PL/SQL Inquerires
- ACCESSIBLE BY Clause
- More PL/SQL-Only Data Types Can Cross PL/SQL-to-SQL Interface
- Invoker's Rights Functions Can Be Result-Cached
- New procedure in DBMS_UTILITY
- New Package: UTL_CALL_STACK
- PL/SQL Functions in SQL statements

ACCESSIBLE BY clause I.

```
CREATE OR REPLACE FUNCTION TAX(P_AMOUNT IN NUMBER)
RETURN NUMBER
ACCESSIBLE BY (depts,scott.depts2)
IS
M  NUMBER;
BEGIN
IF p_amount <8000 THEN
M:=0.08;
ELSIF p_amount <18000 THEN
M:=0.25;
ELSE
M:=0.31;
END IF;
RETURN P_AMOUNT*M;
END;
/
GRANT EXECUTE ON tax TO scott;
```

ACCESSIBLE BY clause II.

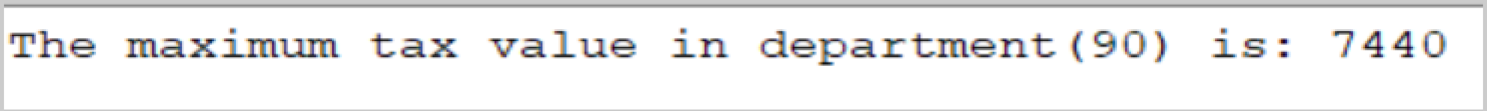
```
CREATE OR REPLACE PROCEDURE depts(p_deptno NUMBER)  IS
summary NUMBER:=0;
v_dept_name departments.department_name%TYPE;
BEGIN
SELECT SUM(salary) INTO summary
FROM employees WHERE department_id=p_deptno;
SELECT department_name INTO v_dept_name
FROM departments WHERE department_id=p_deptno;
dbms_output.put_line
('Total salary for  '||v_dept_name||': '||summary);
EXCEPTION
WHEN no_data_found THEN
dbms_output.put_line('No department !');
END depts;
/
EXEC depts(90)
```



```
Total salary for  Executive: 58000
```

ACCESSIBLE BY clause III.
(working as SCOTT)

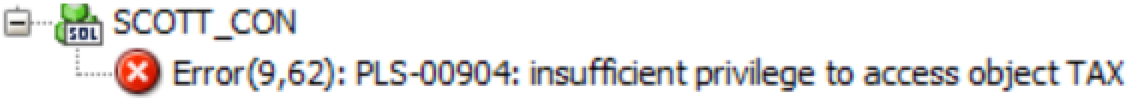
```
CREATE OR REPLACE PROCEDURE
depts2(p_deptno NUMBER:=90)
IS
v_max_sal NUMBER;
BEGIN
SELECT MAX(salary) INTO v_max_sal
FROM HR.employees
WHERE department_id = p_deptno;
dbms_output.put_line
('The maximum tax value in department('||p_deptno||') is: '||hr.tax(v_max_sal));
END depts2;
/
EXEC DEPTS2(90)
```



```
The maximum tax value in department(90) is: 7440
```

ACCESSIBLE BY clause IV.
(working as SCOTT, but!)

```
CREATE OR REPLACE PROCEDURE
depts3(p_deptno NUMBER:=90)
IS
v_max_sal NUMBER;
BEGIN
SELECT MAX(salary) INTO v_max_sal
FROM HR.employees
WHERE department_id = p_deptno;
dbms_output.put_line
('The maximum tax value in department('||p_deptno||') is: '||hr.tax(v_max_sal));
END depts3;
/
```



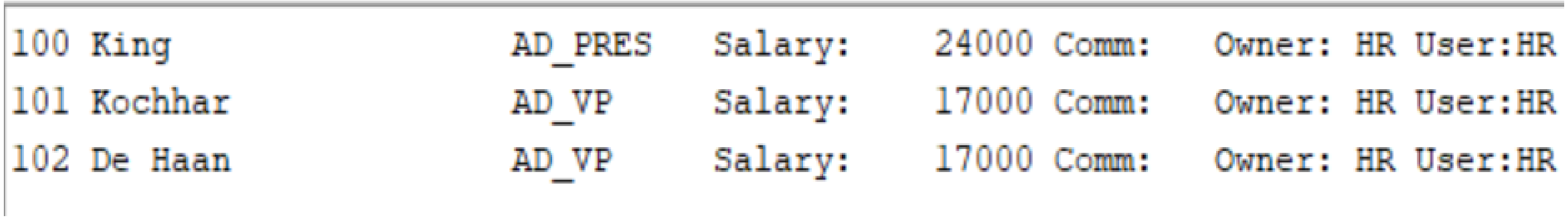
```
SCOTT_CON
  Error(9,62): PLS-00904: insufficient privilege to access object TAX
```

Using Inqueries in Oracle 12c $$PLSQL_UNIT_OWNER with Procedure

```
CREATE OR REPLACE PROCEDURE workers_result_set_p( p_filter VARCHAR2 )
IS
c_emp SYS_REFCURSOR;
r employees%rowtype;
BEGIN
OPEN c_emp FOR
  'SELECT *   FROM employees WHERE '||p_filter;
LOOP
  FETCH c_emp INTO R;
  EXIT WHEN c_emp%notfound;
  dbms_output.put_line(  R.employee_id||' '||rpad(R.last_name,20,' ')
  ||' '||rpad(R.job_id,10,' ')||
  (CASE WHEN $$plsql_unit_owner =USER  THEN
   'Salary: '||lpad(R.salary,8,' ')||' Comm:
    '||to_char(R.commission_pct,'9.99')
   ELSE    ' ' END )
   || ' ' ||' Owner: '||$$PLSQL_UNIT_OWNER||' User:'||user);
    END LOOP;
END workers_result_set_p;
/
```

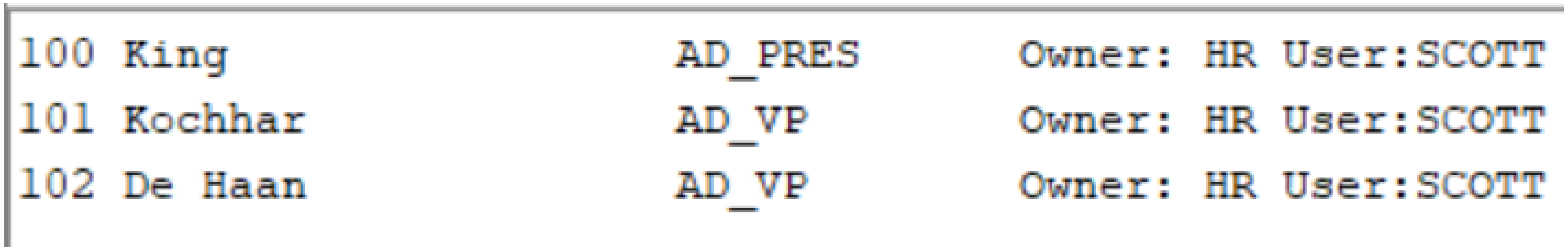
Using Inqueries in Oracle 12c
$$PLSQL_UNIT_OWNER

```
GRANT execute on workers_result_set_p TO scott;
EXEC workers_result_set_p('department_id=90')
```



```
100 King                AD_PRES   Salary:    24000 Comm:   Owner: HR User:HR
101 Kochhar             AD_VP     Salary:    17000 Comm:   Owner: HR User:HR
102 De Haan             AD_VP     Salary:    17000 Comm:   Owner: HR User:HR
```

```
--as scott
SET SERVEROUTPUT ON
exec hr.workers_result_set_p('department_id=90')
```



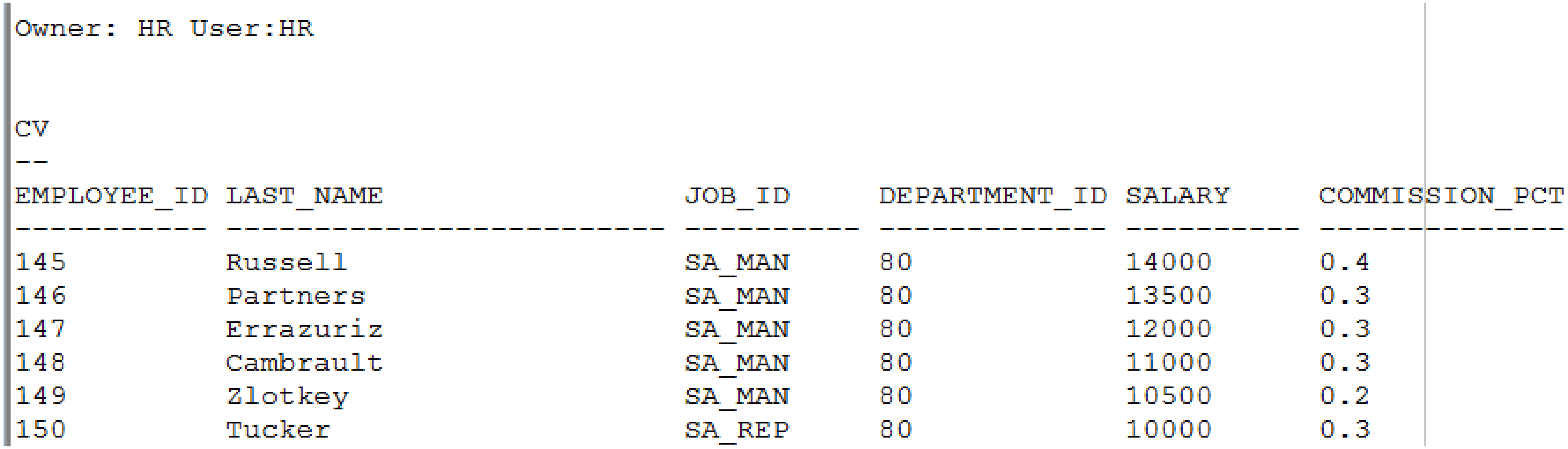
```
100 King                AD_PRES     Owner: HR User:SCOTT
101 Kochhar             AD_VP       Owner: HR User:SCOTT
102 De Haan             AD_VP       Owner: HR User:SCOTT
```

Using Inqueries in Oracle 12c
$$PLSQL_UNIT_OWNER with PL/SQL Function

```
CREATE OR REPLACE FUNCTION employees_result_set
( p_filter VARCHAR2 )
RETURN sys_refcursor
IS
retval sys_refcursor;
v_sens_cols VARCHAR2(300);
BEGIN
dbms_output.put_line
('Owner: '||$$PLSQL_UNIT_OWNER||' User:'||user);
v_sens_cols:=CASE WHEN $$PLSQL_UNIT_OWNER =USER
THEN ', salary, commission_pct '    ELSE ' ' END;
OPEN retval FOR  'SELECT employee_id
    ,last_name,job_id,department_id '||   v_sens_cols||
     ' FROM employees WHERE '||p_filter;
RETURN retval;
END employees_result_set;
/
```

Execute the function as HR

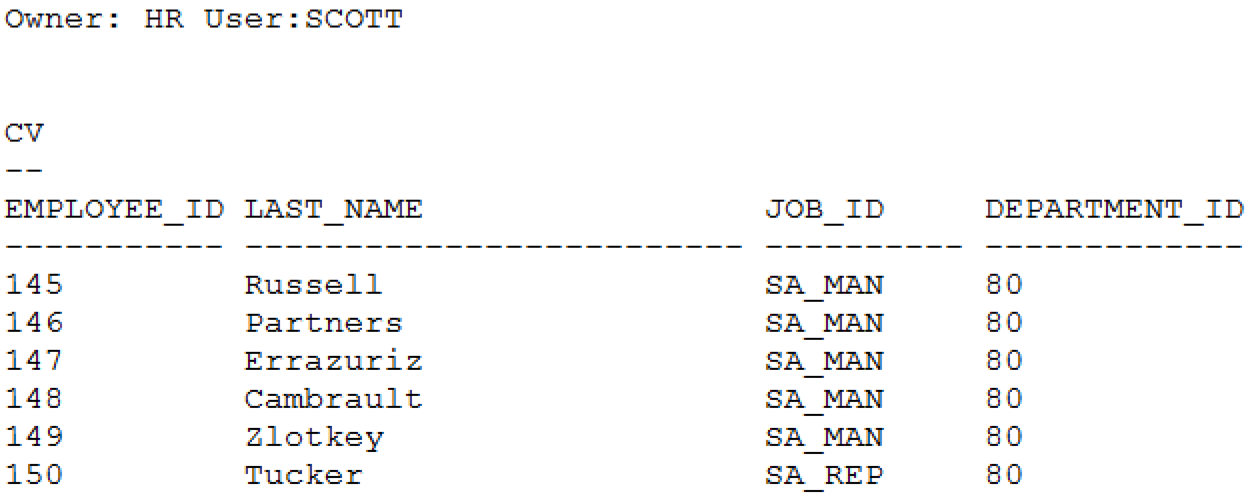
```
GRANT EXECUTE ON employees_result_set TO scott;
VAR cv refcursor
exec :cv:=employees_result_set('department_id=80')
col last_name format a16
PRINT CV
```



```
Owner: HR User:HR


CV
--
EMPLOYEE_ID LAST_NAME                 JOB_ID     DEPARTMENT_ID SALARY     COMMISSION_PCT
----------- ------------------------- ---------- ------------- ---------- --------------
145         Russell                   SA_MAN     80            14000      0.4
146         Partners                  SA_MAN     80            13500      0.3
147         Errazuriz                 SA_MAN     80            12000      0.3
148         Cambrault                 SA_MAN     80            11000      0.3
149         Zlotkey                   SA_MAN     80            10500      0.2
150         Tucker                    SA_REP     80            10000      0.3
```

Execute the function as SCOTT

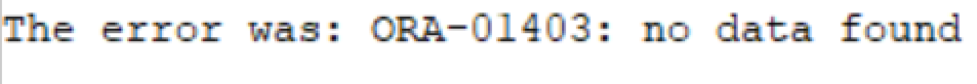
```
VAR cv refcursor
exec :cv:=hr.employees_result_set('department_id=80')
col last_name format a16
PRINT CV
```



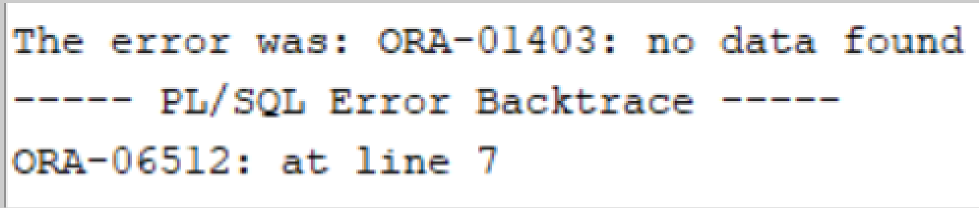
```
Owner: HR User:SCOTT


CV
--
EMPLOYEE_ID LAST_NAME                 JOB_ID     DEPARTMENT_ID
----------- ------------------------- ---------- -------------
145         Russell                   SA_MAN     80
146         Partners                  SA_MAN     80
147         Errazuriz                 SA_MAN     80
148         Cambrault                 SA_MAN     80
149         Zlotkey                   SA_MAN     80
150         Tucker                    SA_REP     80
```

Handling Exceptions: Simple example

```
DECLARE
w employees%ROWTYPE;
m employees%ROWTYPE;
d departments%ROWTYPE;
BEGIN
SELECT * INTO w FROM employees   WHERE employee_id = 100;
SELECT * INTO m FROM employees   WHERE employee_id  =w.manager_id; --7.
SELECT * INTO d FROM departments WHERE department_id=w.department_id;
DBMS_OUTPUT.PUT_LINE
(w.last_name||','||m.last_name||','||d.department_name);
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE('The error was: '||SQLERRM);
error_back_trace;
END;
/
```



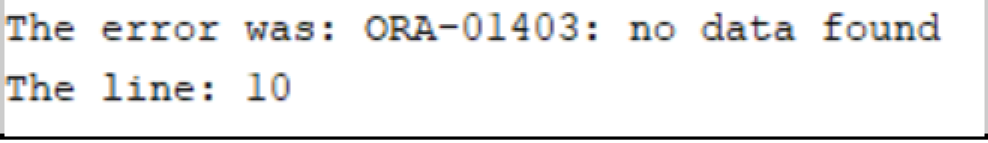
```
The error was: ORA-01403: no data found
```



```
The error was: ORA-01403: no data found
----- PL/SQL Error Backtrace -----
ORA-06512: at line 7
```

Handling Exceptions: $$PLSQL_LINE

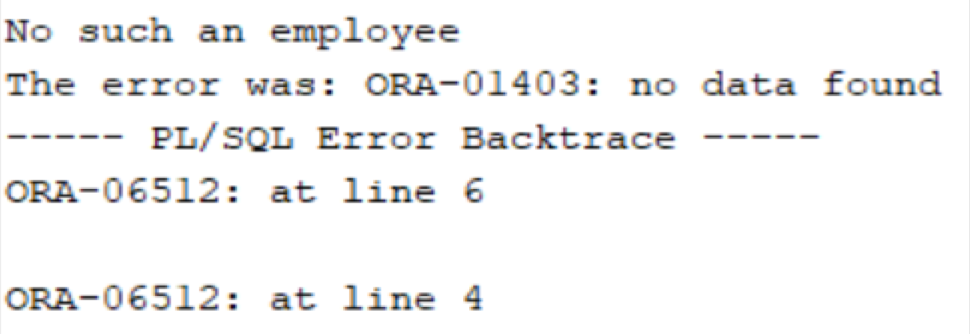
```
DECLARE
w employees%ROWTYPE;
m employees%ROWTYPE;
d departments%ROWTYPE;
stmt_line pls_integer;
BEGIN
stmt_line := $$plsql_line+1;
SELECT * INTO w FROM employees   WHERE employee_id  =&empno; --100
stmt_line := $$PLSQL_LINE+1;
SELECT * INTO m FROM employees   WHERE employee_id  =w.manager_id;--10.
stmt_line := $$PLSQL_LINE+1;
SELECT * INTO d FROM departments WHERE department_id=w.department_id;
dbms_output.put_line(w.last_name||','||m.last_name||','||d.department_n
ame);
EXCEPTION
WHEN no_data_found THEN
dbms_output.put_line('The error was: '||sqlerrm);
dbms_output.put_line('The line: '||stmt_line);
END;
/
```



```
The error was: ORA-01403: no data found
The line: 10
```

Handling Exceptions: Good example

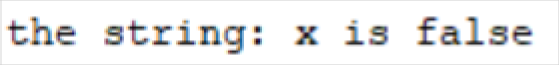
```
DECLARE w employees%ROWTYPE; m employees%ROWTYPE; d departments%ROWTYPE;
BEGIN
  BEGIN
  SELECT * INTO w FROM employees WHERE employee_id=&empno;
  EXCEPTION WHEN NO_DATA_FOUND THEN
  DBMS_OUTPUT.PUT_LINE('No such an employee'); RAISE;
  END;
  BEGIN
  SELECT * INTO m FROM employees WHERE employee_id=w.manager_id;
  EXCEPTION WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('No manager!');
  END;
  BEGIN
  SELECT * INTO d FROM departments WHERE department_id=w.department_id;
  EXCEPTION WHEN NO_DATA_FOUND THEN DBMS_OUTPUT.PUT_LINE('No department!');
  END;
DBMS_OUTPUT.PUT_LINE
(w.last_name||','||m.last_name||','||d.department_name);
EXCEPTION
WHEN NO_DATA_FOUND THEN
DBMS_OUTPUT.PUT_LINE
('The error was: '||SQLERRM);
error_back_trace;
END;
/
```



```
No such an employee
The error was: ORA-01403: no data found
------ PL/SQL Error Backtrace ------
ORA-06512: at line 6

ORA-06512: at line 4
```

More PL/SQL-Only Data Types Can Cross PL/SQL-to-SQL InterfaceI

```
CREATE OR REPLACE FUNCTION p(x boolean) RETURN VARCHAR2 IS
BEGIN
IF x THEN   RETURN 'x is true';
    ELSE RETURN 'x is false';
END IF;
END;
/
set serveroutput on
DECLARE
 l boolean:=5=6;
 s varchar2(30);
 begin
SELECT p(l) INTO s FROM dual;
 dbms_output.put_line('the string: '||s);
END;
 /
```

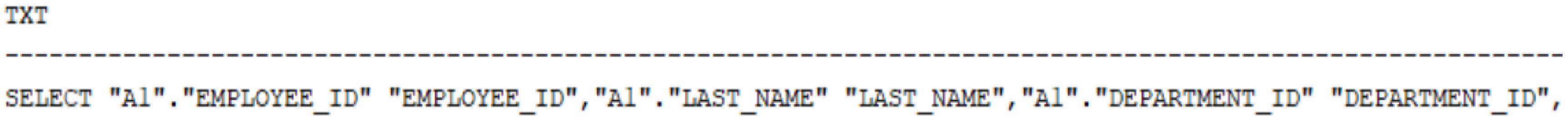


```
the string: x is false
```

New procedure in DBMS_UTILITY EXPAND_SQL_TEXT

- Recursively replaces any view references in the input SQL query with the corresponding view subquery

```
CREATE OR REPLACE VIEW ed  AS
SELECT e.employee_id, e.last_name,
d.department_id, d.department_name
FROM employees E, departments d
WHERE e.employee_id = d.department_id;

SELECT  * FROM  ed;
VAR txt VARCHAR2(500)
SET AUTOPRINT ON
EXEC  DBMS_UTILITY.EXPAND_SQL_TEXT ('SELECT * FROM ed',:txt)
```



```
TXT
----------------------------------------------------------------------------------------------------------
SELECT "A1"."EMPLOYEE_ID" "EMPLOYEE_ID","A1"."LAST_NAME" "LAST_NAME","A1"."DEPARTMENT_ID" "DEPARTMENT_ID",
```

Using UTL_CALL_STACK (Tom Kyte’s demo) http://tkyte.blogspot.hu/2013/06/12c-utlcallstack.html

```
CREATE OR REPLACE PROCEDURE CALLING AS
Depth pls_integer := UTL_Call_Stack.Dynamic_Depth(); d pls_integer:=0;
PROCEDURE headers  is
     begin
        dbms_output.put_line( 'Depth             Number              Name ' );
        dbms_output.put_line( '----------           -----------    ------------------------' );
    end headers;
BEGIN
DBMS_Output.Put_Line('Depth:'||Depth||chr(10)); headers;
for j in reverse 1..Depth loop
    d:=d+1;
    DBMS_Output.Put_Line(
    lpad( utl_call_stack.lexical_depth(j), 10 ) ||rpad( d, 7) ||
    lpad( To_Char(UTL_Call_Stack.Unit_Line(j), '99'), 9 ) ||
   lpad(UTL_Call_Stack.Concatenate_Subprogram(UTL_Call_Stack.Subprogram(j)),30,' '));
end loop;
END CALLING;
/
```

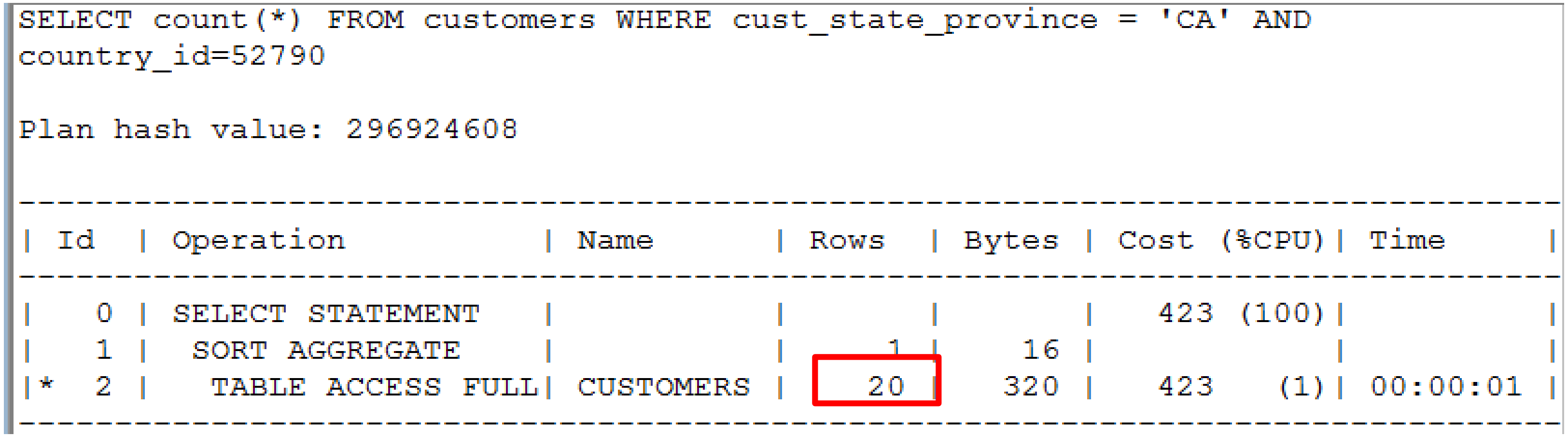
Column Statistics: Extended Statistics

- The optimizer poorly estimates selectivity on *Highly Correlated Column Predicates:*
  - Columns have values that are highly correlated.
  - Actual selectivity is often much lower or higher than the optimizer estimates. For example,

    ```
    WHERE cust_state_province = 'CA'
    AND country_id=52790;
    ```
- The optimizer poorly estimates *Expression on Columns:*
  - `WHERE upper(model)='MODEL'`
  - When a function is applied to a column in the `WHERE` clause, the optimizer has no way of knowing how that function affects the selectivity of the column.

Example for extended statistics

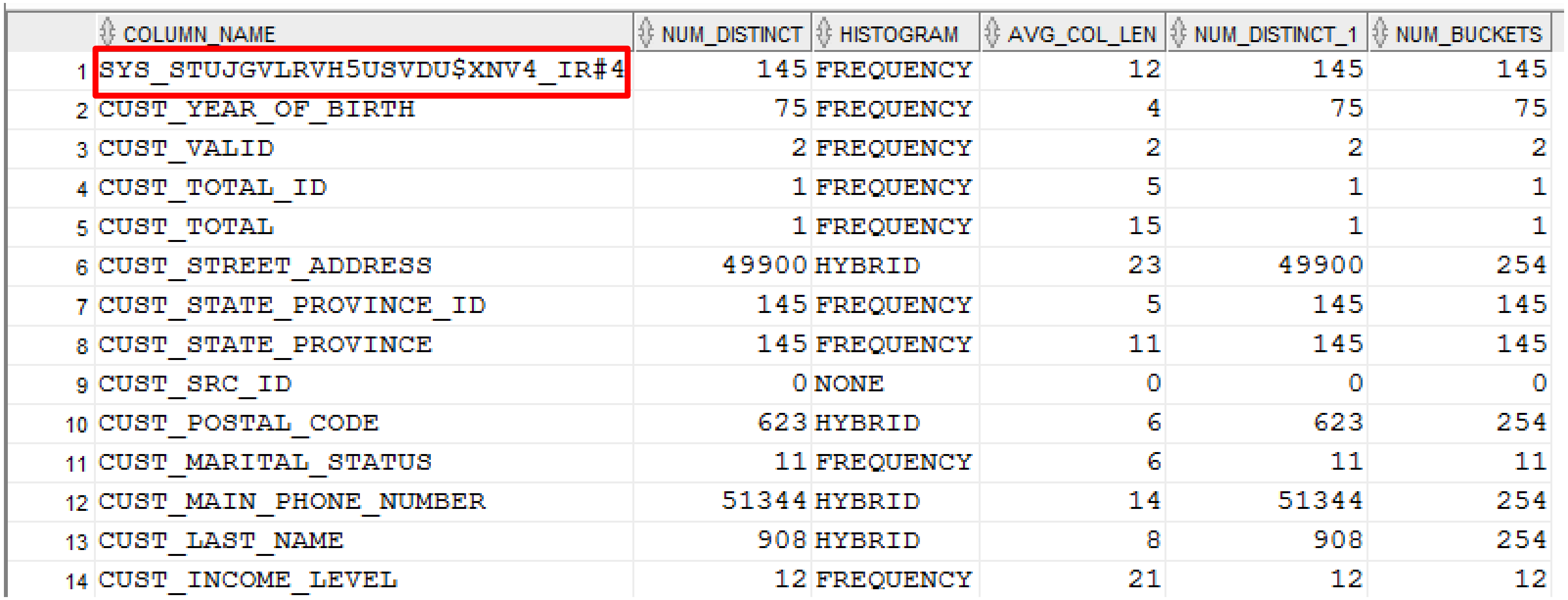
```
SELECT count(*) FROM customers
WHERE cust_state_province = 'CA' AND country_id=52790;
 COUNT(*)
----------
      3341
```



```
SELECT count(*) FROM customers WHERE cust_state_province = 'CA' AND
country_id=52790

Plan hash value: 296924608

--------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows   | Bytes | Cost (%CPU)| Time       |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |        |       |   423 (100)|            |
|   1 |  SORT AGGREGATE     |             |      1 |    16 |            |            |
|*  2 |   TABLE ACCESS FULL| CUSTOMERS   |     20 |   320 |   423   (1)| 00:00:01 |
--------------------------------------------------------------------------------
```

Example for extended statistics (CL)

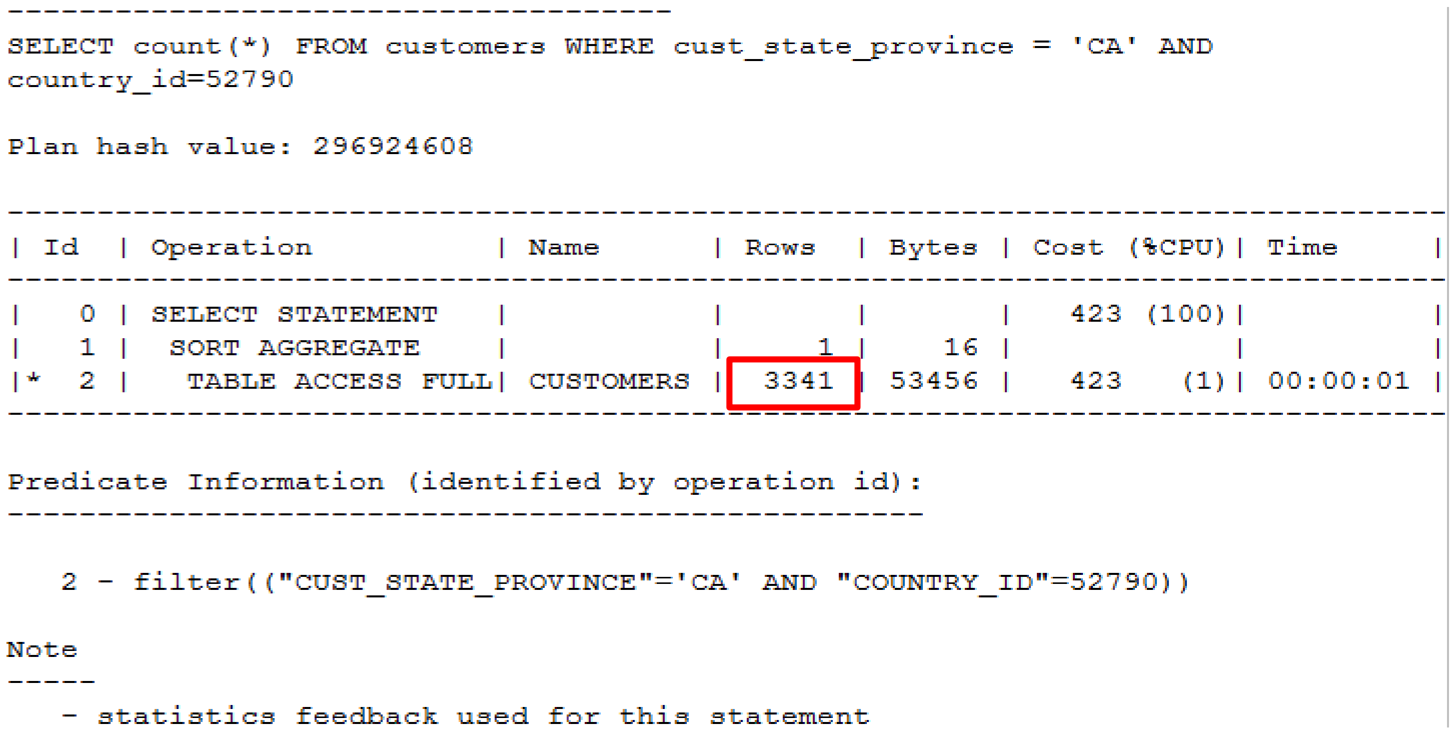
```
SELECT  dbms_stats.create_extended_stats(NULL,'customers',
'(country_id, cust_state_province)') from dual;

SELECT column_name, num_distinct, histogram,
avg_col_len,num_distinct,num_buckets
FROM user_tab_col_statistics WHERE table_name = 'CUSTOMERS'
ORDER BY column_name DESC;
```



	COLUMN_NAME	NUM_DISTINCT	HISTOGRAM	AVG_COL_LEN	NUM_DISTINCT_1	NUM_BUCKETS
1	SYS_STUJGVLRVH5USVDU$XNV4_IR#4	145	FREQUENCY	12	145	145
2	CUST_YEAR_OF_BIRTH	75	FREQUENCY	4	75	75
3	CUST_VALID	2	FREQUENCY	2	2	2
4	CUST_TOTAL_ID	1	FREQUENCY	5	1	1
5	CUST_TOTAL	1	FREQUENCY	15	1	1
6	CUST_STREET_ADDRESS	49900	HYBRID	23	49900	254
7	CUST_STATE_PROVINCE_ID	145	FREQUENCY	5	145	145
8	CUST_STATE_PROVINCE	145	FREQUENCY	11	145	145
9	CUST_SRC_ID	0	NONE	0	0	0
10	CUST_POSTAL_CODE	623	HYBRID	6	623	254
11	CUST_MARITAL_STATUS	11	FREQUENCY	6	11	11
12	CUST_MAIN_PHONE_NUMBER	51344	HYBRID	14	51344	254
13	CUST_LAST_NAME	908	HYBRID	8	908	254
14	CUST_INCOME_LEVEL	12	FREQUENCY	21	12	12

Example for extended statistics (CL)

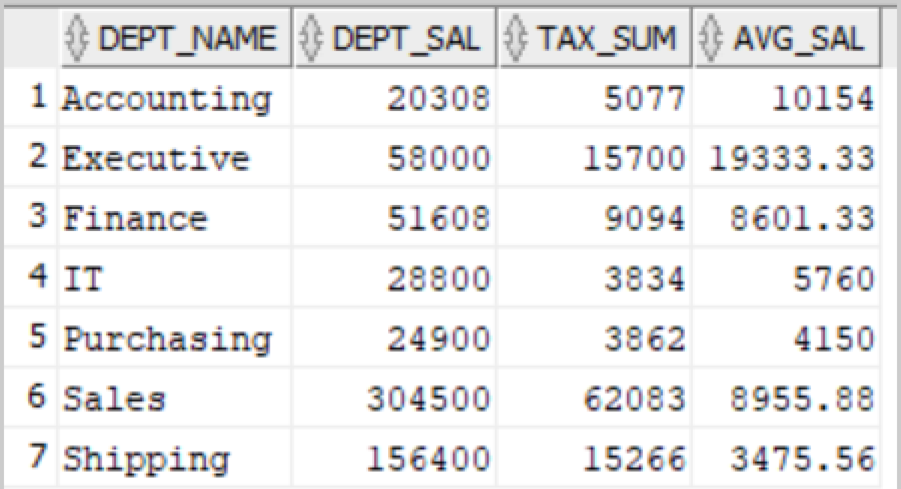
```
Exec dbms_stats.gather_table_stats(null,'customers',
     method_opt => 'for all columns size skewonly');
SELECT count(*) FROM customers
WHERE cust_state_province = 'CA' AND country_id=52790;
```



```
--------------------------------------
SELECT count(*) FROM customers WHERE cust_state_province = 'CA' AND
country_id=52790

Plan hash value: 296924608

--------------------------------------------------------------------------------
| Id  | Operation            | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |           |       |       |   423 (100)|          |
|   1 |  SORT AGGREGATE      |           |     1 |    16 |            |          |
|*  2 |   TABLE ACCESS FULL| CUSTOMERS |  3341 | 53456 |   423   (1)| 00:00:01 |
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter(("CUST_STATE_PROVINCE"='CA' AND "COUNTRY_ID"=52790))

Note
-----
   - statistics feedback used for this statement
```

WITH option using local PL/SQL subprogram I.

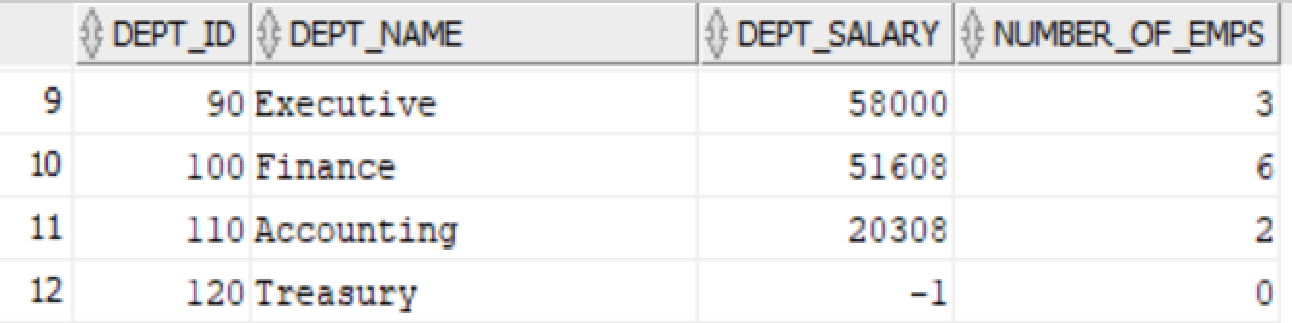
```
WITH FUNCTION tax(p_amount IN NUMBER)
RETURN NUMBER IS
m  NUMBER;
BEGIN
IF p_amount <8000 THEN m:=0.08;
ELSIF p_amount <18000 THEN m:=0.25;
ELSE  m:=0.3;
END IF;
RETURN p_amount * m;
END;
emp_costs  AS (    SELECT d.department_name dept_name,e.last_name,
    e.salary, tax(e.salary) AS tax_amount
    FROM    employees e JOIN departments d    ON    e.department_id = d.department_id),
dept_costs AS (    SELECT dept_name, SUM(salary) AS dept_sal,
    SUM(tax_amount) tax_sum, ROUND(AVG(salary),2) avg_sal
    FROM    emp_costs GROUP BY dept_name)
 SELECT *  FROM    dept_costs
 WHERE  dept_sal >    (SELECT MAX(avg_sal) FROM dept_costs)
 ORDER BY dept_name;
```



	DEPT_NAME	DEPT_SAL	TAX_SUM	AVG_SAL
1	Accounting	20308	5077	10154
2	Executive	58000	15700	19333.33
3	Finance	51608	9094	8601.33
4	IT	28800	3834	5760
5	Purchasing	24900	3862	4150
6	Sales	304500	62083	8955.88
7	Shipping	156400	15266	3475.56

WITH option using local PL/SQL subprogram II.

```
WITH
FUNCTION  dept_sal(p_deptno employees.department_id%TYPE)
RETURN NUMBER IS
summa NUMBER;
BEGIN
SELECT SUM(salary) INTO summa FROM employees
WHERE department_id=p_deptno;
IF summa IS NULL THEN
  RETURN -1;
ELSE
  RETURN summa;
END IF;
END dept_sal;
emp_costs  AS (
SELECT department_id dept_id,department_name dept_name,
(SELECT COUNT(*) FROM employees E WHERE E.department_id=D.department_id)
 number_of_emps,  dept_sal(d.department_id) AS dept_salary
FROM     departments D )
SELECT dept_id, dept_name, dept_salary,number_of_emps
FROM    emp_costs;
```



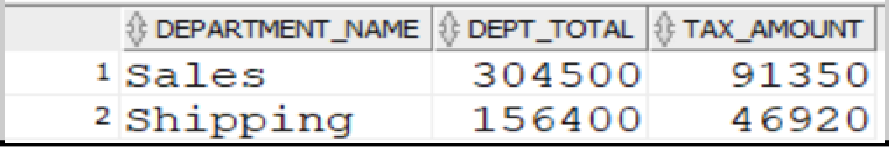
	DEPT_ID	DEPT_NAME	DEPT_SALARY	NUMBER_OF_EMPS
9	90	Executive	58000	3
10	100	Finance	51608	6
11	110	Accounting	20308	2
12	120	Treasury	-1	0

Using PL/SQL function in UPDATE Statement

```
DROP TABLE NEWEMP PURGE;
CREATE TABLE newemp AS SELECT  * FROM employees;
ALTER TABLE  newemp ADD tax_amount number(10,2);
UPDATE  /*+ WITH_PLSQL */ newemp E
SET tax_amount=(WITH FUNCTION TAX(P_AMOUNT IN NUMBER)
RETURN NUMBER  IS
M  NUMBER;
BEGIN
IF P_AMOUNT <8000 THEN M:=0.08;
ELSIF P_AMOUNT <18000 THEN M:=0.25;
ELSE  M:=0.3;
END IF;
RETURN P_AMOUNT*M;
END;
SELECT tax(salary) FROM employees M
WHERE m.employee_id=e.employee_id);
/
SELECT salary, tax_amount FROM newemp ORDER BY salary;
```

Using PL/SQL function in CREATE VIEW Statement

```
CREATE OR REPLACE VIEW proba  As
WITH FUNCTION TAX(P_AMOUNT IN NUMBER)
RETURN NUMBER IS M  NUMBER;
BEGIN
IF P_AMOUNT <8000 THEN M:=0.08;
ELSIF P_AMOUNT <18000 THEN M:=0.25;
ELSE M:=0.3;
END IF;
RETURN P_AMOUNT*M;
END;
dept_costs  AS (    SELECT d.department_name, SUM(e.salary) dept_total,
  TAX(SUM(E.SALARY)) TAX_AMOUNT
   FROM   employees e JOIN departments d    ON    e.department_id = d.department_id
   GROUP BY d.department_name),
 avg_cost    AS (    SELECT  AVG(dept_total) dept_avg, SUM(TAX_AMOUNT) TAX
    FROM   dept_costs)
 SELECT *  FROM   dept_costs   WHERE  dept_total >
(SELECT dept_avg   FROM avg_cost)
ORDER BY department_name;
```



	DEPARTMENT_NAME	DEPT_TOTAL	TAX_AMOUNT
1	Sales	304500	91350
2	Shipping	156400	46920

Köszönöm a figyelmet!

Czinkóczki László

laszlo. czinkoczki@webvalto.hu