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

laszlo.czinkoczki@webvalto.hu
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

```sql
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;
```

Also:

```sql
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
TABLESPACE hr_emp;
```
Using identity in CREATE TABLE statement

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

```sql
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(DESCRIPTIOM) VALUES ('HOUG');
INSERT INTO identity_test_tab(DESCRIPTIOM) 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;
```

![Sequence Table](image)
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.

Statistics Collector: NL
Valid range: 1 to 100
Join will switch to HJ if 100 rows or more are read.
Rows are buffered up to that point.
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.

```
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;
```

```
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> select * from table(dbms_xplan.display());
PLAN_TABLE_OUTPUT
Plan hash value: 389188998

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>NESTED LOOPS</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS FULL</td>
<td>ORDER_ITEMS</td>
</tr>
<tr>
<td>4</td>
<td>INDEX UNIQUE SCAN</td>
<td>PRODUCT_INFORMATION_PK</td>
</tr>
<tr>
<td>5</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PRODUCT_INFORMATION</td>
</tr>
</tbody>
</table>
```
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: 000007F03815530 | Hash value: 1077417386

SQL ID q6ts80t03h5da, 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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>E-Rows</th>
<th>OMem</th>
<th>1Mem</th>
<th>O/1/M</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>HASH JOIN</td>
<td></td>
<td>13</td>
<td>2061K</td>
<td>2061K</td>
<td>1/0/0</td>
</tr>
<tr>
<td>-</td>
<td>NESTED LOOPS</td>
<td></td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>NESTED LOOPS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>STATISTICS COLLECTOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>TABLE ACCESS FULL</td>
<td>ORDER_ITEMS</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>INDEX UNIQUE SCAN</td>
<td>PRODUCT_INFORMATION_PK</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>PRODUCT_INFORMATION</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>TABLE ACCESS FULL</td>
<td>PRODUCT_INFORMATION</td>
<td>288</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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;
```
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;
```
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.

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

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

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

```
SQL_ID   dz7cpwllyf25w5, 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   9ld9nbp2rcqy3, child number 0
--------------
SELECT /* HOUG2019 */ * FROM xhun ORDER BY nlssort(name, 'NLS_SORT = Xhungarian')

Plan hash value: 1294398657
```
The NTH_VALUE Function

```sql
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;
```
The APPROX functions in Oracle 12c R2

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

```sql
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
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

```sql
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
  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;
```

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>START_TSTAMP</th>
<th>BOTTOM_TSTAMP</th>
<th>END_TSTAMP</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>05-APR-2011</td>
<td>06-APR-2011</td>
<td>10-APR-2011</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>14-APR-2011</td>
<td>16-APR-2011</td>
<td>18-APR-2011</td>
<td>24</td>
</tr>
</tbody>
</table>
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;

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>TSTAMP</th>
<th>MATCH_NUM</th>
<th>VAR_MATCH</th>
<th>START_TSTAMP</th>
<th>END_TSTAMP</th>
<th>PRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME</td>
<td>05-APR-11</td>
<td>1</td>
<td>STRT</td>
<td>05-APR-11</td>
<td>13-APR-11</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>06-APR-11</td>
<td>1</td>
<td>DOWN</td>
<td>05-APR-11</td>
<td>13-APR-11</td>
<td>12</td>
</tr>
<tr>
<td>ACME</td>
<td>07-APR-11</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>13-APR-11</td>
<td>15</td>
</tr>
<tr>
<td>ACME</td>
<td>08-APR-11</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>13-APR-11</td>
<td>20</td>
</tr>
<tr>
<td>ACME</td>
<td>09-APR-11</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>13-APR-11</td>
<td>24</td>
</tr>
<tr>
<td>ACME</td>
<td>10-APR-11</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>13-APR-11</td>
<td>25</td>
</tr>
<tr>
<td>ACME</td>
<td>11-APR-11</td>
<td>1</td>
<td>DOWN</td>
<td>05-APR-11</td>
<td>13-APR-11</td>
<td>19</td>
</tr>
<tr>
<td>ACME</td>
<td>12-APR-11</td>
<td>1</td>
<td>DOWN</td>
<td>05-APR-11</td>
<td>13-APR-11</td>
<td>15</td>
</tr>
<tr>
<td>ACME</td>
<td>13-APR-11</td>
<td>1</td>
<td>UP</td>
<td>05-APR-11</td>
<td>13-APR-11</td>
<td>25</td>
</tr>
</tbody>
</table>
Example for **ALL ROWS PER MATCH** in Oracle BI EE (Direct Access)
Example for ALL ROWS PER MATCH in Oracle BI EE (Graph with Line type)
Example for W shape

```sql
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;
```
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.

```sql
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 HIERARCHY

- 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.

```sql
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;
```
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 AS caption,
       amount_sold, sales_cal_year_ago AS 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;

<table>
<thead>
<tr>
<th>TIME</th>
<th>PRODUCT</th>
<th>CUSTOMER</th>
<th>CAPTION</th>
<th>AMOUNT_SOLD</th>
<th>YEAR_AGO</th>
<th>PCTCHG_CAL_YEAR_AGO</th>
</tr>
</thead>
<tbody>
<tr>
<td>11998</td>
<td>Boys</td>
<td>Americas</td>
<td>Region name:Americas</td>
<td>5970.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21998</td>
<td>Women</td>
<td>Europe</td>
<td>Region name:Europe</td>
<td>44622.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31998</td>
<td>Girls</td>
<td>Europe</td>
<td>Region name:Europe</td>
<td>6207.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41998</td>
<td>Boys</td>
<td>Europe</td>
<td>Region name:Europe</td>
<td>11544.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51998</td>
<td>Men</td>
<td>Europe</td>
<td>Region name:Europe</td>
<td>38958.65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>61998</td>
<td>Girls</td>
<td>Europe</td>
<td>Region name:Americas</td>
<td>7346.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>71998</td>
<td>Men</td>
<td>Americas</td>
<td>Region name:Americas</td>
<td>19855.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>81998</td>
<td>Women</td>
<td>Americas</td>
<td>Region name:Americas</td>
<td>37141.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91998</td>
<td>Boys</td>
<td>Americas</td>
<td>Region name:Americas</td>
<td>3356.35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101999</td>
<td>Girls</td>
<td>Europe</td>
<td>Region name:Europe</td>
<td>13733.9</td>
<td>6207.5</td>
<td>1.21</td>
</tr>
</tbody>
</table>
### Sales Calendar Half Year Ago

```sql
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;
```

<table>
<thead>
<tr>
<th>TIME</th>
<th>PRODUCT</th>
<th>CUSTOMER</th>
<th>AMOUNT SOLD</th>
<th>SALES_CAL_YEAR_AGO</th>
<th>SALES_CAL_QUARTERS_AGO</th>
<th>SALES_PCTCHG_CAL_YEAR_AGO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998-Q1 Boys Americas 649</td>
<td>Americas</td>
<td>289.8</td>
<td></td>
<td>4530.5</td>
<td>289.8</td>
<td>1.2</td>
</tr>
<tr>
<td>1998-Q2 Boys Americas 4530.5</td>
<td>Americas</td>
<td>501</td>
<td>649</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1998-Q3 Boys Americas 1429.35</td>
<td>Americas</td>
<td>1494</td>
<td>649</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1998-Q4 Boys Americas</td>
<td>Americas</td>
<td>174</td>
<td>4530.5</td>
<td>1429.35</td>
<td></td>
<td>-0.96</td>
</tr>
<tr>
<td>1999-Q1 Boys Americas</td>
<td>Americas</td>
<td>259</td>
<td>1494</td>
<td>174</td>
<td></td>
<td>-0.48</td>
</tr>
<tr>
<td>1999-Q2 Boys Americas</td>
<td>Americas</td>
<td>2325</td>
<td>1429.35</td>
<td>749</td>
<td></td>
<td>1.33</td>
</tr>
<tr>
<td>1999-Q3 Boys Americas</td>
<td>Americas</td>
<td>3325</td>
<td>4927.9</td>
<td>1494</td>
<td></td>
<td>2.3</td>
</tr>
<tr>
<td>1999-Q4 Boys Americas</td>
<td>Americas</td>
<td>983.8</td>
<td>174</td>
<td>749</td>
<td></td>
<td>9.25</td>
</tr>
<tr>
<td>2000-Q1 Boys Americas</td>
<td>Americas</td>
<td>3796</td>
<td>259</td>
<td></td>
<td></td>
<td>13.66</td>
</tr>
</tbody>
</table>
Sales Calendar Quarter Ago with ROLLUP operator

```
SELECT TIME, product,
       SUM(amount_sold), SUM(sales_cal_year_ago) AS year_ago,
       SUM(sales_cal_quarters_ago) AS 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);
```
Some useful DD views

```sql
SELECT * FROM user_attribute_dimensions;
```

<table>
<thead>
<tr>
<th>DIMENSION_NAME</th>
<th>DIMENSION_TYPE</th>
<th>ALL_MEMBER_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH_TIMES_ATTR_DIM</td>
<td>STANDARD</td>
<td>'ALL YEARS'</td>
</tr>
<tr>
<td>SH_PRODUCTS_ATTR_DIM</td>
<td>STANDARD</td>
<td>'ALL PRODUCTS'</td>
</tr>
<tr>
<td>SH_CUSTOMERS_ATTR_DIM</td>
<td>STANDARD</td>
<td>'ALL CUSTOMERS'</td>
</tr>
<tr>
<td>SH_CHANNELS_ATTR_DIM</td>
<td>STANDARD</td>
<td>'ALL CHANNELS'</td>
</tr>
<tr>
<td>SH_PROMOTIONS_ATTR_DIM</td>
<td>STANDARD</td>
<td>'ALL PROMOTIONS'</td>
</tr>
</tbody>
</table>

```sql
SELECT * FROM user_hierarchies;
```

<table>
<thead>
<tr>
<th>HIER_NAME</th>
<th>DIMENSION_OWNER</th>
<th>DIMENSION_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH_TIMES_CALENDAR_HIER</td>
<td>SH_AV</td>
<td>SH_TIMES_ATTR_DIM</td>
</tr>
<tr>
<td>SH_PRODUCTS_HIER</td>
<td>SH_AV</td>
<td>SH_PRODUCTS_ATTR_DIM</td>
</tr>
<tr>
<td>SH_CUSTOMERS_HIER</td>
<td>SH_AV</td>
<td>SH_CUSTOMERS_ATTR_DIM</td>
</tr>
<tr>
<td>SH_CHANNELS_HIER</td>
<td>SH_AV</td>
<td>SH_CHANNELS_ATTR_DIM</td>
</tr>
<tr>
<td>SH_PROMOTIONS_HIER</td>
<td>SH_AV</td>
<td>SH_PROMOTIONS_ATTR_DIM</td>
</tr>
<tr>
<td>SH_TIMES_FISCAL_HIER</td>
<td>SH_AV</td>
<td>SH_TIMES_ATTR_DIM</td>
</tr>
</tbody>
</table>

```sql
SELECT * FROM user_ANALYTIC_views;
```

<table>
<thead>
<tr>
<th>ANALYTIC_VIEW_NAME</th>
<th>TABLE_OWNER</th>
<th>TABLE_NAME</th>
<th>TABLE_ALIAS</th>
<th>DEFAULT_AGGR</th>
<th>DEFAULT_MEASURE</th>
<th>COMPILE_STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH_SALES_HISTORY_AV_2_YEARS</td>
<td>SH_AV</td>
<td>SALES</td>
<td>SALES</td>
<td>SUM</td>
<td>AMOUNT_SOLD</td>
<td>VALID</td>
</tr>
<tr>
<td>SH_SALES_HISTORY_AV_QTR</td>
<td>SH_AV</td>
<td>SALES</td>
<td>SALES</td>
<td>SUM</td>
<td>AMOUNT_SOLD</td>
<td>VALID</td>
</tr>
<tr>
<td>SH_SALES_HISTORY_AV</td>
<td>SH_AV</td>
<td>SALES</td>
<td>SALES</td>
<td>SUM</td>
<td>AMOUNT_SOLD</td>
<td>VALID</td>
</tr>
</tbody>
</table>
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
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;
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
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
depths3(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;
/

$ORACLEROAD$
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 with Procedure
Using Inqueries in Oracle 12c

**$$PLSQL[Unit][Owner]**

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>King</strong></td>
<td><strong>AD_PRES</strong></td>
<td>Salary:</td>
<td>24000</td>
<td>Comm:</td>
<td>Owner:</td>
<td>HR User:</td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Kochhar</strong></td>
<td><strong>AD_VP</strong></td>
<td>Salary:</td>
<td>17000</td>
<td>Comm:</td>
<td>Owner:</td>
<td>HR User:</td>
</tr>
<tr>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>De Haan</strong></td>
<td><strong>AD_VP</strong></td>
<td>Salary:</td>
<td>17000</td>
<td>Comm:</td>
<td>Owner:</td>
<td>HR User:</td>
</tr>
<tr>
<td>102</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

```
--as scott
SET SERVEROUTPUT ON
exec hr.workers_result_set_p('department_id=90')
```
Using Inqueries in Oracle 12c

```sql
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
```
Execute the function as SCOTT

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

-------- PL/SQL Error Backtrace --------
ORA-06512: at line 7
Handling Exceptions: $$PLSQL\_LINE$$

```sql
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;--101
    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_name);
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
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); 
beg
cnt 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)
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;
/

Using UTL_CALL_STACK (Tom Kyte’s demo)
http://tkyte.blogspot.hu/2013/06/12c-utlcallstack.html
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,
    \[
    \text{WHERE cust_state_province = 'CA' AND country_id=52790;}
    \]

- The optimizer poorly estimates *Expression on Columns*:
  - \[
    \text{WHERE upper(model)='MODEL'}
    \]
  - When a function is applied to a column in the \textbf{WHERE} clause, the optimizer has no way of knowing how that function affects the selectivity of the column.
Example for extended statistics

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

```sql
SELECT count(*) FROM customers WHERE cust_state_province = 'CA' AND country_id=52790
Plan hash value: 296924608

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>423</td>
<td>423 (100)</td>
<td>00:00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>16</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>CUSTOMERS</td>
<td>20</td>
<td>320</td>
<td>423 (1)</td>
<td></td>
</tr>
</tbody>
</table>
```
Example for extended statistics (CL)

```sql
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;
```

<table>
<thead>
<tr>
<th>COLUMN_NAME</th>
<th>NUM_DISTINCT</th>
<th>HISTOGRAM</th>
<th>AVG_COL_LEN</th>
<th>NUM_DISTINCT_1</th>
<th>NUM_BUCKETS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_STUJGVRVH5USVDUSXNV4_IR#4</td>
<td>145</td>
<td>FREQUENCY</td>
<td>12</td>
<td>145</td>
<td>145</td>
</tr>
<tr>
<td>CUST_YEAR_OF_BIRTH</td>
<td>75</td>
<td>FREQUENCY</td>
<td>4</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td>CUST_VALID</td>
<td>2</td>
<td>FREQUENCY</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>CUST_TOTAL_ID</td>
<td>1</td>
<td>FREQUENCY</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CUST_TOTAL</td>
<td>1</td>
<td>FREQUENCY</td>
<td>15</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CUST_STREET_ADDRESS</td>
<td>49900 HYBRID</td>
<td>23</td>
<td>49900</td>
<td>254</td>
<td>254</td>
</tr>
<tr>
<td>CUST_STATE_province_ID</td>
<td>145</td>
<td>FREQUENCY</td>
<td>5</td>
<td>145</td>
<td>145</td>
</tr>
<tr>
<td>CUST_STATE_PROVINCE</td>
<td>145</td>
<td>FREQUENCY</td>
<td>11</td>
<td>145</td>
<td>145</td>
</tr>
<tr>
<td>CUST_SRC_ID</td>
<td>0</td>
<td>NONE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CUST_POSTAL_CODE</td>
<td>623</td>
<td>HYBRID</td>
<td>6</td>
<td>623</td>
<td>254</td>
</tr>
<tr>
<td>CUST_MARITAL_STATUS</td>
<td>11</td>
<td>FREQUENCY</td>
<td>6</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>CUST_MAIN_PHONE_NUMBER</td>
<td>51344 HYBRID</td>
<td>14</td>
<td>51344</td>
<td>254</td>
<td>254</td>
</tr>
<tr>
<td>CUST_LAST_NAME</td>
<td>908</td>
<td>HYBRID</td>
<td>8</td>
<td>908</td>
<td>254</td>
</tr>
<tr>
<td>CUST_INCOME_LEVEL</td>
<td>12</td>
<td>FREQUENCY</td>
<td>21</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>423 (100)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>CUSTOMERS</td>
<td>3341</td>
<td>53456</td>
<td>423 (1)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>

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

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

Note
-----
- statistics feedback used for this statement
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),
depth_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;
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;
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;
```

<table>
<thead>
<tr>
<th>DEPARTMENT_NAME</th>
<th>DEPT_TOTAL</th>
<th>TAX_AMOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>304500</td>
<td>91350</td>
</tr>
<tr>
<td>Shipping</td>
<td>156400</td>
<td>46920</td>
</tr>
</tbody>
</table>
Köszönöm a figyelmet!

Czinkóczki László
laszlo.czinkoczki@webvalto.hu