Lesson-03 :: Group Functions, Joins, Sub-Queries

Group Functions

AKA - Aggregate Functions
Multi-row Functions

They work on a group of rows.
If the grouping is not explicitly specified, all the rows of the table are taken into account.

Examples

```
SQL> SELECT SUM( salary ) FROM employees;

SUM(SALARY)
-----------
     691416

SQL> SELECT AVG( salary ) FROM employees;

AVG(SALARY)
-----------
 6461.83178

SQL> SELECT MIN( salary ) FROM employees;

MIN(SALARY)
-----------
       2100

SQL> SELECT MAX( salary ) FROM employees;

MAX(SALARY)
-----------
      24000


SQL> SELECT COUNT( commission_pct ) FROM employees;

COUNT(COMMISSION_PCT)
---------------------
                   36
```

NOTE: With COUNT() if a column name is specified, it returns the count of Non-NULL values.

```
SQL> SELECT COUNT( * ) FROM employees;

  COUNT(*)
----------
       107
```

Using GROUP BY Clause

The GROUP BY clause enables grouping of rows.

```
SQL> SELECT department_id, SUM( salary ) FROM employees
  2  GROUP BY department_id
  3  ORDER BY department_id;

DEPARTMENT_ID SUM(SALARY)
------------- -----------
           10        4400
           20       19000
           30       24900
           40        6500
           50      156400
           60       28800
           70       10000
           80      304500
           90       58000
          100       51608
          110       20308


SQL> SELECT department_id, SUM( salary ), AVG( salary ), MIN( salary ),
MAX( salary ) FROM employees
  2  GROUP BY department_id
  3  ORDER BY department_id;

DEPARTMENT_ID SUM(SALARY) AVG(SALARY) MIN(SALARY) MAX(SALARY)
------------- ----------- ----------- ----------- -----------
           10        4400        4400        4400        4400
           20       19000        9500        6000       13000
           30       24900        4150        2500       11000
           40        6500        6500        6500        6500
           50      156400  3475.55556        2100        8200
           60       28800        5760        4200        9000
           70       10000       10000       10000       10000
           80      304500  8955.88235        6100       14000
           90       58000  19333.3333       17000       24000
          100       51608  8601.33333        6900       12008
          110       20308       10154        8300       12008
                     7000        7000        7000        7000


The HAVING Clause
=================
Used to filter group(s) based on a condition.

  1  SELECT department_id, SUM( salary ), AVG( salary ), MIN( salary ),
MAX( salary ) FROM employees
  2  GROUP BY department_id
  3  HAVING AVG(salary) >= 10000
  4* ORDER BY department_id
SQL> /

DEPARTMENT_ID SUM(SALARY) AVG(SALARY) MIN(SALARY) MAX(SALARY)
------------- ----------- ----------- ----------- -----------
           70       10000       10000       10000       10000
           90       58000  19333.3333       17000       24000
          110       20308       10154        8300       12008
```

Example:

```
SQL> -- Getting a HEAD COUNT department-wise
SQL> ed
Wrote file afiedt.buf

  1  SELECT department_id, COUNT( * ) "No of Employees" FROM employees
  2  GROUP BY department_id
  3* ORDER BY 1
SQL> /

DEPARTMENT_ID No of Employees
------------- ---------------
           10               1
           20               2
           30               6
           40               1
           50              45
           60               5
           70               1
           80              34
           90               3
          100               6
          110               2
```

Get the group which has employees more than or equal to 30

```
SQL> ed
Wrote file afiedt.buf

  1  SELECT department_id, COUNT( * ) "No of Employees" FROM employees
  2  GROUP BY department_id
  3  HAVING COUNT( * ) >= 30
  4* ORDER BY 1
SQL> /

DEPARTMENT_ID No of Employees
------------- ---------------
           50              45
           80              34
```

Getting a count of employees based on the job id. Moreover the count should be greater than or equal to FIVE.

```
  1  SELECT job_id, COUNT( * ) FROM employees
  2  GROUP BY job_id
  3* HAVING COUNT( * ) >= 5
SQL> /

JOB_ID       COUNT(*)
---------- ----------
FI_ACCOUNT          5
IT_PROG             5
PU_CLERK            5
SA_MAN              5
SA_REP             30
SH_CLERK           20
ST_CLERK           20
ST_MAN              5
```

Example:
Getting a count of 'first_name' which occurs more than once

```
  1  SELECT first_name, COUNT( * ) FROM employees
  2  GROUP BY first_name
  3* HAVING COUNT(*) > 1
SQL> /

FIRST_NAME             COUNT(*)
-------------------- ----------
Peter                         3
Michael                       2
Steven                        2
John                          3
Julia                         2
William                       2
Karen                         2
Kevin                         2
David                         3
Jennifer                      2
Randall                       2
```

Example:
Getting a count of 'first_name' more than once, but only from department 50 or 80

```
  1  SELECT first_name, COUNT( * ) FROM employees
  2  WHERE department_id IN (50, 80)
  3  GROUP BY first_name
  4* HAVING COUNT(*) > 1
SQL> /

FIRST_NAME             COUNT(*)
-------------------- ----------
Peter                         3
Julia                         2
John                          2
Kevin                         2
Randall                       2
David                         2
James                         2
```

##

Why split data in Multiple-Tables?
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
RDBMS Concepts
- Normalization
- Minimizing Data Redundancy
- Data Consistency
We deliberately split the data in multiple tables.

What is Joining?
==========
Joining is ability of the SELECT statement to access the data existing in two or more tables.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Oracle supports
- Oracle proprietary joins
- SQL 1999 compliant joins

Cartesian Product

It combines the number of rows of first table with the number of rows of the second table and gives a complete list.

i.e. If Table-A has 5 rows and Table-B has 3 row, the Cartesian Product will generate 15 rows.

This will happen when we do not have a VALID where clause.

Example-1: Oracle proprietary

```
SQL> SELECT department_id, city
  2  FROM departments, locations;
```

NOTE: The WHERE clause is NOT SPECIFIED in this case.

Example-2: Using SQL 1999 compliant

```
SELECT department_id, city
FROM departments
CROSS JOIN locations;
```

Types of Joins
---
[1] Equi-Join
    We make use of the EQUALITY Operator (i.e. =)
[2] Outer Join
    Right Outer Join
    Left Outer Join
    Full Outer Join
[3] Non-Equi Join
    We make use of Operator other than the Equality (i.e. =)
[4] Self Join
    If a table joins to itself it is called Self Join
    It is also called as Recursive Relationship.

EquiJoin
===
Example-1a

```
SQL> SELECT department_name, city
  2  FROM   departments, locations
  3  WHERE departments.location_id = locations.location_id;

DEPARTMENT_NAME                CITY
------------------------------ ------------------------------
IT                             Southlake
Shipping                       South San Francisco
Administration                 Seattle
Purchasing                     Seattle
```

```
:
:
Payroll                        Seattle
Marketing                      Toronto
Human Resources                London
Sales                          Oxford
Public Relations               Munich

Example-1b : SQL 1999 Compliant
SQL> ed
Wrote file afiedt.buf

  1  SELECT department_name, city
  2  FROM departments
  3* NATURAL JOIN locations


Example-2a - Getting the employee first name and department name

SQL> SELECT first_name, department_name
  2  FROM employees, departments
  3  WHERE employees.department_id = departments.department_id;

FIRST_NAME           DEPARTMENT_NAME
-------------------- ------------------------------
Jennifer             Administration
Pat                  Marketing
Michael              Marketing
Sigal                Purchasing

Example-2b - Getting the employee first name and department name


Example-3a : Getting employee first name and job title
SQL> SELECT first_name, job_title
  2  FROM employees, jobs
  3  WHERE employees.job_id = jobs.job_id;

FIRST_NAME           JOB_TITLE
-------------------- ----------------------------------
William              Public Accountant
Shelley              Accounting Manager
Jennifer             Administration Assistant
Steven               President
Neena                Administration Vice President
Lex                  Administration Vice President
Jose Manuel          Accountant
Ismael               Accountant
:
:

Example-3b : Getting employee first name and job title

SELECT first_name, job_title
FROM employees
NATURAL JOIN jobs
```

Non-Equi Join
==========
Example-1

```
SQL> SELECT ename, sal, grade
  2  FROM emp, salgrade
  3  WHERE sal >= losal AND sal <= hisal;

ENAME             SAL      GRADE
---------- ---------- ----------
SMITH              800          1
JAMES              950          1
ADAMS             1100          1
WARD              1250          2
MARTIN            1250          2
MILLER            1300          2
TURNER            1500          3
ALLEN             1600          3
CLARK             2450          4
SCOTT             3000          4
FORD              3000          4
KING              5000          5

  1  SELECT ename, sal, grade
  2  FROM emp, salgrade
  3* WHERE sal BETWEEN losal AND hisal;
```

Outer Join
==========

Example

```
SQL> SELECT ename, dname FROM emp, dept
  2  WHERE emp.deptno = dept.deptno;

ENAME      DNAME
---------- --------------
CLARK      ACCOUNTING
KING       ACCOUNTING
MILLER     ACCOUNTING
JONES      RESEARCH
FORD       RESEARCH
ADAMS      RESEARCH
SMITH      RESEARCH
SCOTT      RESEARCH
WARD       SALES
TURNER     SALES
ALLEN      SALES
JAMES      SALES
BLAKE      SALES
MARTIN     SALES

14 rows selected.
```

However, there is a department by name OPERATIONS for which there are no employees.

To get the department name for which there are no employees we use the OUTER JOIN concept.

Example-1a

```
  1  SELECT ename, dname FROM emp, dept
  2* WHERE emp.deptno(+) = dept.deptno;
SQL> /

ENAME      DNAME
---------- --------------
CLARK      ACCOUNTING
KING       ACCOUNTING
MILLER     ACCOUNTING
JONES      RESEARCH
FORD       RESEARCH
ADAMS      RESEARCH
SMITH      RESEARCH
SCOTT      RESEARCH
WARD       SALES
TURNER     SALES
ALLEN      SALES
JAMES      SALES
BLAKE      SALES
MARTIN     SALES
           OPERATIONS
```

NOTE: The (+) syntax for getting Outer Join is Oracle property.

Example-1b : Performing RIGHT OUTER JOIN

```
SELECT ename, dname
FROM emp e
RIGHT OUTER JOIN dept d
ON (e.deptno = d.deptno)
```

Self Join
=========
If a table joins to ITSELF then we say we are performing SELF JOIN

```
SQL> -- The employee name and the manager to whom they report
SQL>
SQL> SELECT worker.ename "Employee Name",  manager.ename "Manager"
  2  FROM emp worker, emp manager
  3  WHERE worker.mgr = manager.empno;

Employee N Manager
---------- ----------
FORD       JONES
SCOTT      JONES
TURNER     BLAKE
ALLEN      BLAKE
WARD       BLAKE
JAMES      BLAKE
MARTIN     BLAKE
MILLER     CLARK
ADAMS      SCOTT
BLAKE      KING
JONES      KING
CLARK      KING
```

Performing an LEFT Outer Join to know which employee does not have a Manager

```
  1  SELECT worker.ename "Employee Name",  manager.ename "Manager"
  2  FROM emp worker, emp manager
  3* WHERE worker.mgr = manager.empno(+)
SQL> /

Employee N Manager
---------- ----------
FORD       JONES
SCOTT      JONES
JAMES      BLAKE
TURNER     BLAKE
MARTIN     BLAKE
WARD       BLAKE
ALLEN      BLAKE
MILLER     CLARK
ADAMS      SCOTT
CLARK      KING
BLAKE      KING
JONES      KING
SMITH      FORD
KING

14 rows selected.
```

Using SQL 1999 complaint

```
SELECT worker.ename "Employee", manager.ename "Manager"
FROM emp worker
LEFT OUTER JOIN emp manager
ON ( worker.mgr = manager.empno)
/
```

We can even perform the same with 'HR' schema.

```
SQL> SELECT w.first_name "Employee", m.first_name "Manager"
  2  FROM employees w, employees m
  3  WHERE w.manager_id = m.employee_id;

Employee             Manager
-------------------- --------------------
William              Gerald
Lisa                 Gerald
Sundita              Gerald
Tayler               Gerald
Harrison             Gerald
Elizabeth            Gerald
Alexander            Lex
Clara                Alberto
Mattea               Alberto
David                Alberto
Danielle             Alberto
Amit                 Alberto
Sundar               Alberto
Nandita              Adam
```

```
TJ                    Adam
:
:

  1  SELECT w.first_name "Employee", m.first_name "Manager"
  2  FROM employees w, employees m
  3* WHERE w.manager_id = m.employee_id(+)
SQL> /

Employee              Manager
--------------------- --------------------
Sundita               Gerald
Elizabeth             Gerald
William               Gerald
Tayler                Gerald
Harrison              Gerald
Lisa                  Gerald
Alexander             Lex
Amit                  Alberto
:
:
Alyssa                Eleni
Ellen                 Eleni
Steven

NOTE: Steven does not have a Manager.

SQL 1999 - Join Syntax
======================
[1] Cross Join

    Creates a Cartesian Product

Example

SQL> ed
Wrote file afiedt.buf

  1  SELECT department_id, city
  2  FROM departments
  3* CROSS JOIN locations
SQL> /

[2] Natural Join

    Is the Equi Join

SQL> SELECT department_name, city
  2  FROM departments
  3  NATURAL JOIN locations;

DEPARTMENT_NAME                CITY
------------------------------ ------------------------------
IT                             Southlake
Shipping                       South San Francisco
Administration                 Seattle
Purchasing                     Seattle
:
```

```
:
Payroll                        Seattle
Marketing                      Toronto
Human Resources                London
Sales                          Oxford
Public Relations               Munich

[2-A] Natural Join - Using the USING Clause

SQL> ed
Wrote file afiedt.buf

  1  SELECT department_name, city
  2  FROM departments
  3  JOIN locations
  4* USING(location_id, location_id)
SQL> /

NOTE: The USING clause should be preferred with the column names
      do not match along with their respective data types.

[2-B] Natural Join - Using the ON Clause

SQL> ed
Wrote file afiedt.buf

  1  SELECT department_name, city
  2  FROM departments
  3  JOIN locations
  4* ON (departments.location_id = locations.location_id)
SQL> /

SQL> ed
Wrote file afiedt.buf

  1  SELECT department_name, city
  2  FROM departments
  3  JOIN locations
  4  ON (departments.location_id = locations.location_id)
  5* AND city IN ('Southlake', 'Oxford', 'Toronto', 'Munich')
SQL> /

DEPARTMENT_NAME                CITY
------------------------------ ------------------------------
Public Relations               Munich
Sales                          Oxford
IT                             Southlake
Marketing                      Toronto

[3] Outer Join

SQL> ed
Wrote file afiedt.buf

  1  SELECT ename, dname
  2  FROM emp RIGHT OUTER JOIN dept
  3* ON (emp.deptno = dept.deptno )
SQL> /
```

```
ENAME      DNAME
---------- --------------
CLARK      ACCOUNTING
KING       ACCOUNTING
MILLER     ACCOUNTING
JONES      RESEARCH
FORD       RESEARCH
ADAMS      RESEARCH
SMITH      RESEARCH
SCOTT      RESEARCH
WARD       SALES
TURNER     SALES
ALLEN      SALES
JAMES      SALES
BLAKE      SALES
MARTIN     SALES
           OPERATIONS

Example
SQL> SELECT ename, dname
  2  FROM  new_emp  LEFT OUTER JOIN dept
  3  ON (new_emp.deptno = dept.deptno);

ENAME      DNAME
---------- --------------
MILLER     ACCOUNTING
KING       ACCOUNTING
CLARK      ACCOUNTING
FORD       RESEARCH
ADAMS      RESEARCH
SCOTT      RESEARCH
JONES      RESEARCH
SMITH      RESEARCH
JAMES      SALES
TURNER     SALES
BLAKE      SALES
MARTIN     SALES
WARD       SALES
ALLEN      SALES
Harshan

Example : Full Outer Join

SQL> ed
Wrote file afiedt.buf

  1  SELECT ename, dname
  2  FROM  new_emp  FULL OUTER JOIN dept
  3* ON (new_emp.deptno = dept.deptno)
SQL> /

ENAME      DNAME
---------- --------------
SMITH      RESEARCH
ALLEN      SALES
WARD       SALES
JONES      RESEARCH
```

```
MARTIN     SALES
BLAKE      SALES
CLARK      ACCOUNTING
SCOTT      RESEARCH
KING       ACCOUNTING
TURNER     SALES
ADAMS      RESEARCH
JAMES      SALES
FORD       RESEARCH
MILLER     ACCOUNTING
Harshan
           OPERATIONS
```

Observe the output, we get those employee(s) who are not mapped to a department, plus we get those department(s) for which we do not have employees.

NOTE: The SQL 1999 Syntax works with other RDBMS which support ANSI SQL

SUMMARY:
A join between two tables that return rows that match the join condition and also unmatched rows from left table is LEFT OUTER JOIN

A join between two tables that return rows that match the join condition and unmatched rows from the right table is RIGHT OUTER JOIN

A join between two tables that return rows that match the join condition and returns unmatched rows of both left and right table is a FULL OUTER JOIN

==========

SubQueries

We have the concept of INNER Query and OUTER Query.
The INNER Query (sub-query) is executed and the result of it used by the
OUTER Query (main-query)

Getting a list of employees who's salary is more than that of Abel's salary

```
SQL> SELECT first_name FROM employees
  2  WHERE salary > ( SELECT salary FROM employees WHERE last_name =
'Abel');

FIRST_NAME
--------------------
Steven
Neena
Lex
Nancy
John
Karen
Alberto
Lisa
Michael
Shelley
```

The INNER Query will find Abel's salary and that salary is used as input to the OUTER Query.

Example-2

```
SQL> -- Getting a list of employees who are in 'Finanace' department
SQL>
SQL> SELECT first_name FROM employees
  2  WHERE department_id = ( SELECT department_id FROM departments WHERE
department_name = 'Finance' );

FIRST_NAME
--------------------
Nancy
Daniel
John
Ismael
Jose Manuel
Luis
```

Multi-Level Nesting of Subqueries

==========

We want to find the full name of the employee who has joined the company first.

Example-1 : Doing it step by step

```
-- #1 : Getting the lowest HIRE_DATE
SQL> SELECT MIN(hire_date) FROM employees;

MIN(HIRE_
---------
13-JAN-01

-- #2 : Getting employee ID of the first employee to hire
SQL> SELECT employee_id FROM employees WHERE hire_date = '13-JAN-01';

EMPLOYEE_ID
-----------
        102

-- #3 : Getting the full name of the employee based on the above ID
SQL> SELECT first_name || ' ' || last_name "Full Name"
  2  FROM employees
  3  WHERE employee_id = 102;

Full Name
----------------------------------------------
Lex De Haan
```

The same can be achived by multi-level nesting of queries as follows:

```
SELECT first_name || ' ' || last_name "Full Name"
FROM employees
WHERE employee_id = ( SELECT employee_id FROM employees
                    WHERE hire_date = (SELECT MIN(hire_date) FROM
employees ));
```

What happens when a subquery returns multiple rows?
~~~~~~
We get an ERROR as follows:

Example:
```
SELECT  last_name, salary FROM employees
WHERE salary > ( SELECT salary FROM employees WHERE last_name = 'Smith'
);

WHERE salary > ( SELECT salary FROM employees WHERE last_name = 'Smith' )
                 *
ERROR at line 2:
ORA-01427: single-row subquery returns more than one row
```

To overcome the above error, we need to use MULTI-ROW operators.

Using Multi-Row Operator with SubQueries

Example

```
SQL> ed
Wrote file afiedt.buf

  1  SELECT first_name FROM employees
  2* WHERE salary IN ( SELECT salary FROM employees WHERE first_name =
'Peter')
SQL> /

FIRST_NAME
--------------------
Randall
Martha
Peter
Joshua
James
Karen
Hermann
Harrison
Janette
Peter
Allan
Peter
Daniel
Alexander
```

NOTE: As we have THREE employees with the first name as 'Peter' we get THREE
different salary values.

Thus to get a list of other employees who get a salary similar to that of 'Peter' we use MULTI-ROW operator 'IN' in this case.

Example
~~~~~~

```
SQL> ed
Wrote file afiedt.buf

  1  SELECT first_name FROM employees
  2* WHERE salary NOT IN ( SELECT salary FROM employees WHERE first_name
= 'Peter')
SQL> /
```

To get the list of employees who's salary is not like 'Peter' we can use NOT IN
multi-row operator.

NOTE: Topics on the following are coming up in Advance SQL
- Co-related Subqueries
- EXISTS and NOT EXISTS Operators

Set Operators - TDB