Build COMPETENCY
across your TEAM

SYNERGETICS GET IT RIGHT

Microsoft Partner
Gold Cloud Platform
Silver Learning

RDBMS Fundamentals-2

Google Cloud Platform
Windows Azure

amazon web services
APACHE kafka. A distributed streaming platform
Apache SOFTWARE FOUNDATION
APACHE Spark
Java
hadoop

Module 4

Joins and Unions

Joins

- A **join** is a query that combines rows from two or more tables, views, or materialized views.
- Oracle Database performs a join whenever multiple tables appear in the FROM clause of the query.
- The select list of the query can select any columns from any of these tables.
- If any two of these tables have a column name in common, then you must qualify all references to these columns throughout the query with table names to avoid ambiguity.

There are multiple ways to join tables.

Types of Joins

- Equijoins
- Non equi joins
- Outer joins
- Self-joins

Cartesian Products

A Cartesian product results in a display of all combinations of rows. This is done by omitting the WHERE clause.

Cartesian Products

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition.

Generally: CROSS PRODUCT is **not meaningful** operation

- Select * From department CROSS JOIN location

- SELECT last_name, department_name
  FROM employees
  CROSS JOIN departments ;

- Select * From employees ,departments

Aliases

- Qualifying column names with table names can be very time consuming, particularly if table names are lengthy.
- Table aliases help to keep SQL code smaller, therefore using less memory.
- Use table aliases to simplify queries.
- Use table aliases to improve performance

```
SELECT e.employee_id, e.last_name,
       d.location_id, department_id
FROM   employees e JOIN departments d
USING (department_id) ;
```

Equi joins

- Equi joins are also called simple joins or inner joins
- An equijoin is a join with a join condition containing an equality operator.
- An equijoin combines rows that have equivalent values for the specified columns.

```
SELECT e.employee_id, e.last_name, e.department_id, d.department_id, d.location_id
FROM   employees e, departments d
WHERE  e.department_id = d.department_id;
```

```
SELECT d.department_id, d.department_name, d.location_id, l.city FROM   departments d, locations l
WHERE  d.location_id = l.location_id;
```

Non-equi joins

- Non equi joins is used to return result from two or more tables where exact join is not possible.

```
SELECT a.department_id, a.department_name, b.city
FROM departments a, locations b
WHERE b.location_id BETWEEN 1800 AND 2500
AND a.department_id < 30;
```

Self-Joins

- you can also join a table to itself.
- Contain tables being referenced more than once in the FROM clause.
- In autojoins, the table names result in ambiguity issues.

```
SELECT worker.last_name || ' works for ' || manager.last_name FROM   employees
worker, employees manager WHERE  worker.manager_id =
manager.employee_id ;
```

Outer Join (Oracle-Specific)

You use an outer join to see rows that do not meet the join condition.

```
SELECT table1.column, table2.column FROM table1, table2 WHERE table1.column(+) = table2.column;
SELECT table1.column, table2.column FROM table1, table2 WHERE table1.column = table2.column(+);
```

```
SELECT e.last_name, e.department_id, d.department_name FROM   employees e, departments d
WHERE  e.department_id(+) = d.department_id ;
```

```
SELECT e.last_name, e.department_id, d.department_name FROM   employees e, departments d
WHERE  e.department_id = d.department_id(+) ;
```

Note:
Oracle's join syntax does not have an equivalent for the FULL OUTER JOIN of the SQL:1999–compliant join syntax.

Joins of Three or More Tables

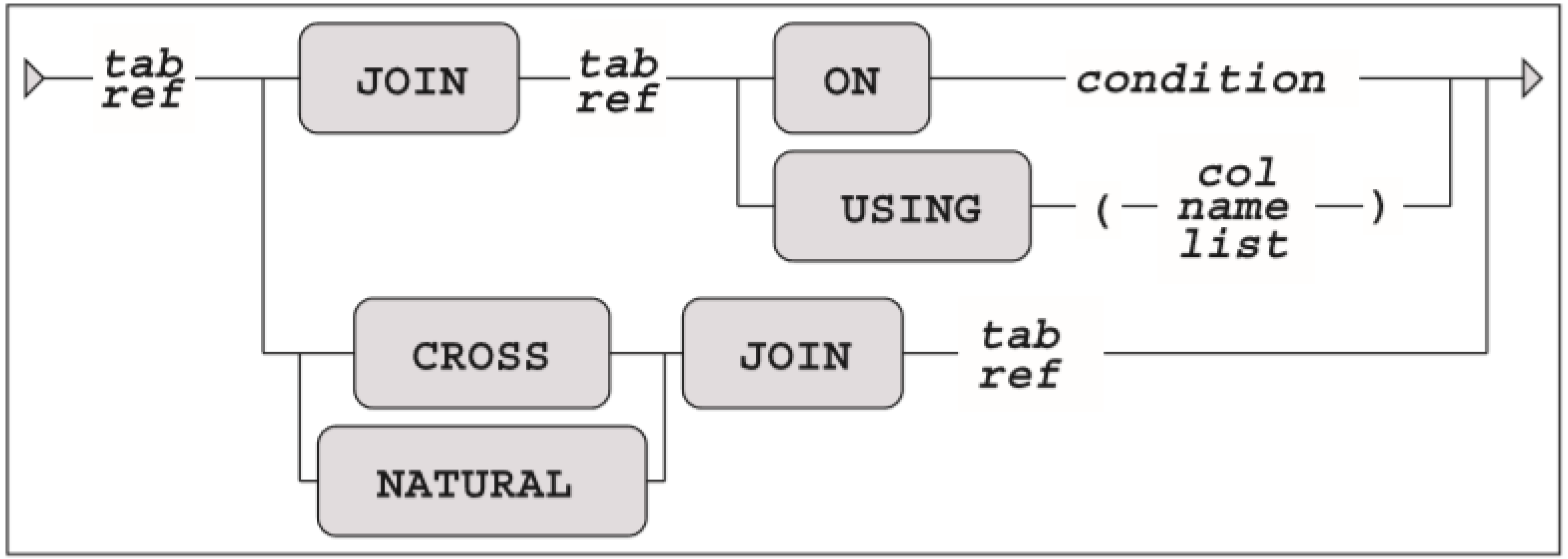
In a three-table join, Oracle joins two of the tables and joins the result with the third table.

```
SELECT e.last_name, d.department_name, l.city FROM   employees e, departments d,
locations l WHERE  e.department_id = d.department_id AND    d.location_id =
l.location_id;
```

```
SELECT e.last_name, e.salary, j.grade_level FROM   employees e, job_grades j WHERE
e.salary BETWEEN j.lowest_sal AND j.highest_sal;
```

Alternative ANSI/ISO Standard Join Syntax (JOIN ON)

ANSI/ISO SQL standard also supports alternative syntax to specify joins



ANSI/ISO join syntax diagram

JOIN and ON.

Also note also that this join syntax doesn’t use any commas in the FROM clause.

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- Use the `ON` clause to specify arbitrary conditions or specify columns to join.
- The join condition is separated from other search conditions.
- The `ON` clause makes code easy to understand.

```
SELECT e.employee_id, e.last_name,
e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id);
```

USING Clause

- If several columns have the same names but the data types do not match, the NATURAL JOIN clause can be modified with the USING clause to specify the columns that should be used for an equijoin.

- Use the USING clause to match only one column when more than one column matches.

```
SELECT employees.employee_id, employees.last_name,
       departments.location_id, department_id
FROM   employees JOIN departments
USING (department_id) ;
```

Natural Joins

The `NATURAL JOIN` clause is based on all columns in the two tables that have the same name.
It selects rows from the two tables that have equal values in all matched columns.
If the columns having the same names have different data types, an error is returned.
Natural join is special case of Equijoin
Natural join removes duplicate attributes

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;
```

You should be very careful when using the NATURAL JOIN operator.
Probably the biggest danger is that a natural join may “suddenly” start producing strange and undesirable results if you add new columns to your tables, or you rename existing columns, thus accidentally creating matching column names.
Caution Natural joins are safe only if you practice a very strict column-naming standard in your database designs

Equi joins on Columns with the Same Name

- SQL offers an alternative way to specify equij oins, allowing you to explicitly specify the columns you want to participate in the equijoin operation.

```
select e.ename, e.bdate
,h.deptno, h.msal
from   employees e
join
history h
using (empno)
where  e.job = 'ADMIN';
```

Outer Joins

```
select d.deptno, d.location
 ,      e.ename, e.init
  from   employees e, departments d
  where  e.deptno = d.deptno
order  by d.deptno, e.ename;
```

- Oracle does not have an equivalent syntax to support the FULL OUTER JOIN of the SQL:1999– compliant join syntax.

Self-Joins Using the `ON` Clause

```
SELECT e.last_name emp, m.last_name mgr
FROM   employees e JOIN employees m
ON    (e.manager_id = m.employee_id);
```

Non Equijoin

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON     e.salary
       BETWEEN j.lowest_sal AND j.highest_sal;
```

Joining More than two table

```
select first_name,department_name,city
from employees JOIN departments using(department_id)
               JOIN locations using(location_id)

select first_name,department_name,city
from employees
JOIN departments
 ON(employees.department_id=departments.department_id)
JOIN locations
 ON(departments.location_id=locations.location_id)
```

LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e LEFT OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

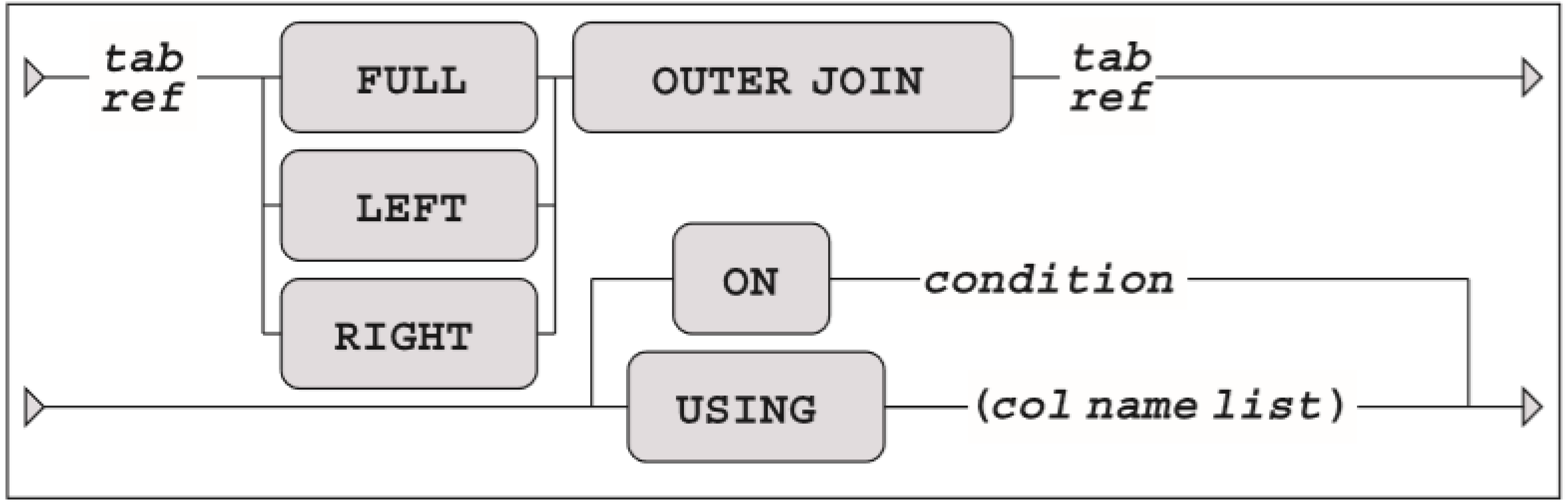
RIGHT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

ANSI/ISO outer join syntax

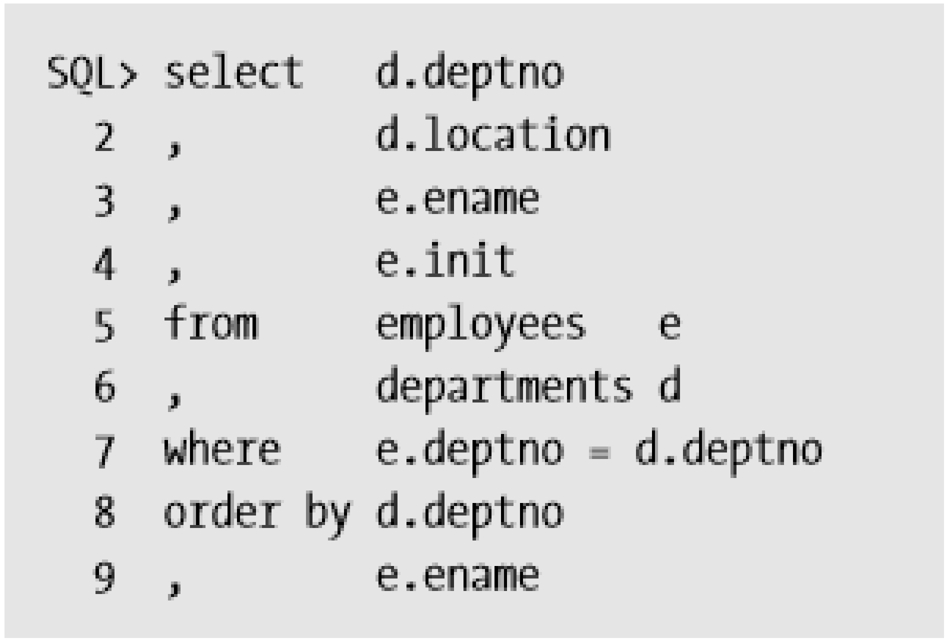


ANSI/ISO outer join syntax diagram

```
select deptno, d.location
,       e.ename, e.init
from    employees e
        right outer join
        departments d
        using (deptno)
order  by deptno, e.ename;
```

Layout for long coded query style.

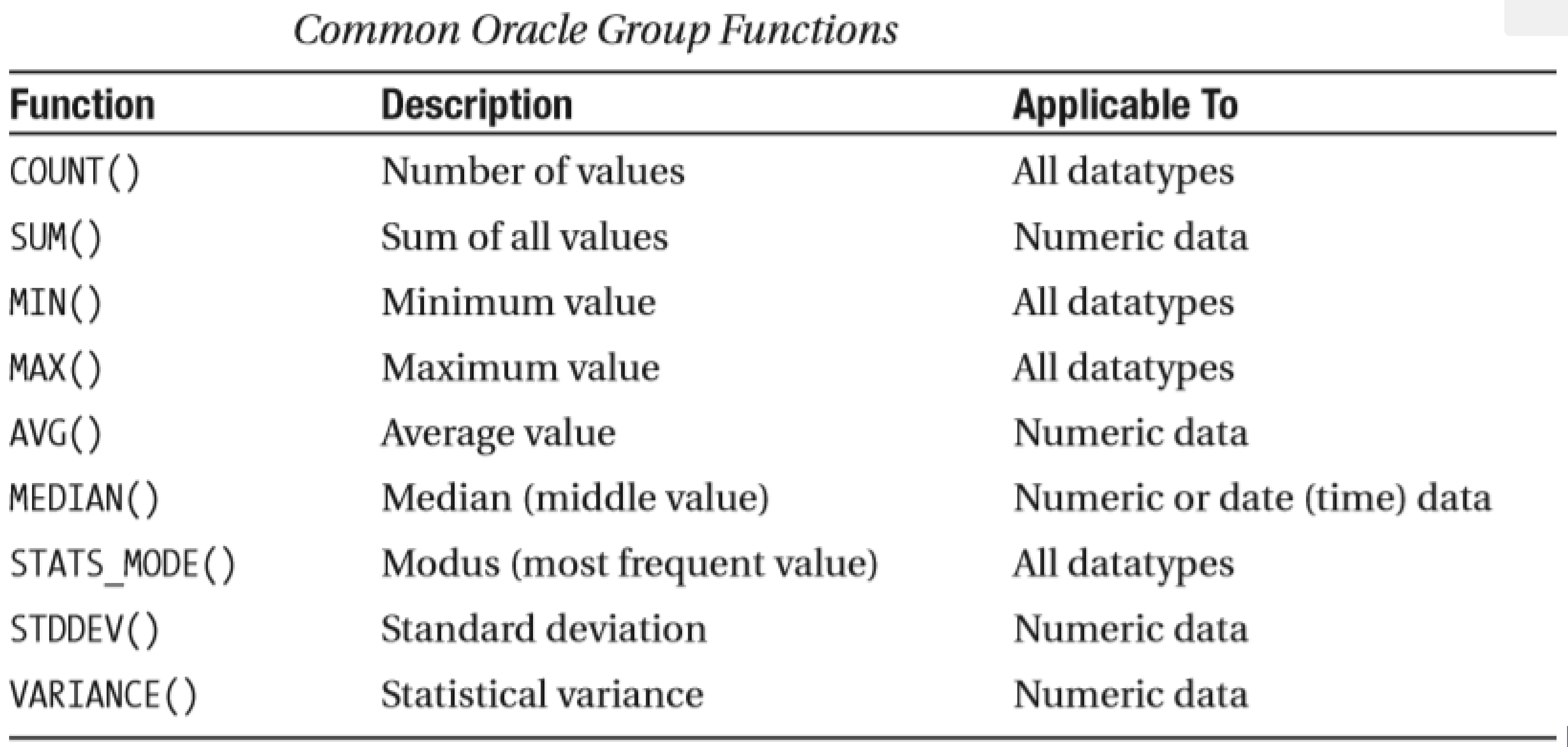
- Your SQL statements should be correct in the first place,of course.
- As soon as SQL statements get longer and more complicated,it becomes more and more important to adopt a certain layout style.
- Additional white space (spaces,tabs,and new lines) has no meaning in the SQL language,but it certainly enhances code readability and maintainability.



```
SQL> select   d.deptno
  2  ,        d.location
  3  ,        e.ename
  4  ,        e.init
  5  from     employees   e
  6  ,        departments d
  7  where    e.deptno = d.deptno
  8  order by d.deptno
  9  ,        e.ename
```

Group Functions

- All group functions have two important properties in common:
  - They can be applied only to sets of values.
  - They return a single aggregated value, derived from that set of values.



Common Oracle Group Functions

Function	Description	Applicable To
`COUNT()`	Number of values	All datatypes
`SUM()`	Sum of all values	Numeric data
`MIN()`	Minimum value	All datatypes
`MAX()`	Maximum value	All datatypes
`AVG()`	Average value	Numeric data
`MEDIAN()`	Median (middle value)	Numeric or date (time) data
`STATS_MODE()`	Modus (most frequent value)	All datatypes
`STDDEV()`	Standard deviation	Numeric data
`VARIANCE()`	Statistical variance	Numeric data

Group BY Syntax

```
SELECT    column, group_function
FROM      table
[WHERE    condition]
[GROUP BY group_by_expression]
[HAVING   group_condition]
[ORDER BY column];
```

GROUP BY

You can divide rows in a table into smaller groups by using the GROUP BY clause.
All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.

```
SELECT   department_id, AVG(salary)
FROM     employees
GROUP BY department_id ;
```

The GROUP BY column does not have to be in the SELECT list.

```
SELECT   AVG(salary)
FROM     employees
GROUP BY department_id ;
```

Aggregate Function

```
SELECT AVG(salary), MAX(salary),
       MIN(salary), SUM(salary)
FROM   employees
```

You can use `MIN` and `MAX` for numeric, character, and date data types.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM   employees;
```

- `COUNT(*)` returns the number of rows in a table:

```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

- `COUNT(`*`expr`*`)` returns the number of rows with non-null values for the *`expr`*:

```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 80;
```

  - `COUNT(DISTINCT` *`expr`*`)` returns the number of distinct non-null values of the *`expr`*.
  - To display the number of distinct department values in the `EMPLOYEES` table:

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```

Null in Aggregate Functions

If a column expression on which you apply the GROUP BY clause contains null values, these null values end up together in a separate group.

```
select e.comm, count(e.empno)
from   employees e
group  by e.comm;
```

```
SELECT AVG(commission_pct)
FROM   employees;
```

- The NVL function forces group functions to include null values:

```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

GROUP BY Clause on Multiple Columns

- Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause:

```
SELECT   department_id dept_id, job_id, SUM(salary)
FROM     employees
GROUP BY department_id, job_id ;

SELECT department_id, COUNT(last_name)
FROM   employees;
ERROR at line 1:
ORA-00937: not a single-group group function
```

Restriction

- You cannot use the `WHERE` clause to restrict groups.
- You use the `HAVING` clause to restrict groups.
- You cannot use group functions in the `WHERE` clause.

```
SELECT   department_id, AVG(salary)
FROM     employees
WHERE    AVG(salary) > 8000
GROUP BY department_id;
ERROR at line 3:
ORA-00934: group function is not allowed here
```

Restrictions

- When you use the `HAVING` clause, the Oracle server restricts groups as follows:
  1. Rows are grouped.
  2. The group function is applied.
  3. Groups matching the `HAVING` clause are displayed.

```
SELECT   department_id, MAX(salary)
FROM     employees
GROUP BY department_id
HAVING   MAX(salary)>10000 ;

SELECT   job_id, SUM(salary) PAYROLL
FROM     employees
WHERE    job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING   SUM(salary) > 13000
ORDER BY SUM(salary);
```

Subqueries

- The subquery (inner query) executes once before the main query (outer query).
- The result of the subquery is used by the main query.

```
SELECT select_list
FROM    table
WHERE   expr operator
         (SELECT select_list
           FROM table);
```

Subquery

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition.
- The `ORDER BY` clause in the subquery is not needed unless you are performing Top-N analysis.
- Use single-row operators with single-row subqueries, and use multiple-row operators with multiple-row subqueries.

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

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

Types of Subqueries

- Single-row subquery
  - Return only one row
  - Use single-row comparison operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

- Multiple-row subquery
  - Return more than one row
  - Use multiple-row comparison operators

Operator	Meaning
IN	Equal to any member in the list
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery

Single-row subquery

- Subqueries that can return only one or zero rows to the outer statement are called **single-row subqueries**.
- Single-row subqueries are subqueries used with a comparison operator in a WHERE, or HAVING clause.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  employee_id = 141)
AND    salary >
                (SELECT salary
                 FROM   employees
                 WHERE  employee_id = 143);
```

Multiple Row Subquery

- Multiple row subquery returns one or more rows to the outer SQL statement.
- You may use the IN, ANY, or ALL operator in outer query to handle a subquery that returns multiple rows.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY
                    (SELECT salary
                     FROM   employees
                     WHERE  job_id = 'IT_PROG')
```

Corelated Subquery

- **A correlated sub query is evaluated once for each row processed by the parent statement (query)**
- Correlated sub queries are used for row-by-row processing.
- A correlated subquery is a subquery that uses values from the outer query.
- The Oracle database wants to execute the subquery once and use the results for all the evaluations in the outer query.
- With a correlated subquery, the database must run the subquery for each evaluation because it is based on the outer query’s data.

```
select ename,job from emp e where EXISTS(select mgr from emp
where(emp.mgr=e.empno));
```

Null Values in a Subquery

- If an inner query returns a NULL, the outer query also returns NULL.
- In an Oracle database, a NULL value means unknown, so any comparison or operation against a NULL value is also NULL, and any test that returns NULL is always ignored

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
                              (SELECT mgr.manager_id
                               FROM   employees mgr);
```

Set Operators

- To combine results from more than one SQL statement
- The SELECT list of each query must match in number and datatype.

- **UNION:** The UNION operator combines the results of more than one SELECT statement after removing any duplicate rows. Oracle will sort the resulting set of data.
- **UNION ALL: S**imilar to UNION, but it doesn't remove the duplicate rows. Oracle doesn't sort the result set in this case, unlike the UNION operation.
- **INTERSECTION: G**ets you the common values in two or more result sets derived from separate SELECT statements. The result set is distinct and sorted.
- **MINUS: R**eturns the rows returned by the first query that aren't in the second query's results. The result set is distinct and sorted.

```
SELECT emp_id FROM old_employees  UNION  SELECT emp_id FROM new_employees;
```

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

Module 5

Insert, Update and Delete operations

Data Manipulation Language (DML)

- A DML statement is executed when you:
  - Add new rows to a table
  - Modify existing rows in a table
  - Remove existing rows from a table
- A *transaction* consists of a collection of DML statements that form a logical unit of work.

I Add new row: (INSERT Statement)

Add new rows to a table by using the `INSERT` statement:

```
INSERT INTO table [(column [, column...])] VALUES
```

With this syntax, only one row is inserted at a time.

- Insert a new row containing values for each column.
- List values in the default order of the columns in the table.
- Optionally, list the columns in the `INSERT` clause.

```
    INSERT INTO departments(department_id,
      department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
```

Inserting Rows with Null Values

```
INSERT INTO   departments
VALUES        (100, 'Finance', NULL, NULL);
```

Inserting Special Values

```
INSERT INTO employees (employee_id,
                 first_name, last_name,
                 email, phone_number,
                 hire_date, job_id, salary,
                 commission_pct, manager_id,
                 department_id)
VALUES           (113,
                 'Louis', 'Popp',
                 'LPOPP', '515.124.4567',
                 SYSDATE, 'AC_ACCOUNT', 6900,
                 NULL, 205, 100);
```

Copying Rows from Another Table

- Write your INSERT statement with a subquery:

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
 SELECT employee_id, last_name, salary, commission_pct
 FROM   employees
 WHERE  job_id LIKE '%REP%';
```

- Do not use the VALUES clause.
- Match the number of columns in the INSERT clause to those in the subquery.

Changing Data in a Table (UPDATE)

```
UPDATE          table
SET        column = value [, column = value, ...]
[WHERE          condition];

UPDATE employees
SET    department_id = 70
WHERE  employee_id = 113;

UPDATE     copy_emp
SET        department_id = 110;
```

Updating Columns with a Subquery

```
UPDATE    employees
SET       job_id   = (SELECT  job_id
                      FROM    employees
                      WHERE   employee_id = 205),
          salary   = (SELECT  salary
                      FROM    employees
                      WHERE   employee_id = 205)
WHERE     employee_id     =  114;
```

Removing a Row from a Table (DELETE)

- You can remove existing rows from a table by using the `DELETE` statement:

```
DELETE [FROM]    table
[WHERE       condition];
DELETE FROM departments
 WHERE  department_name = 'Finance';
```

- All rows in the table are deleted if you omit the `WHERE` clause:

Deleting Rows Based on Another Table

```
DELETE FROM employees
WHERE  department_id =
                 (SELECT department_id
                  FROM   departments
                  WHERE  department_name
                         LIKE '%Public%');
```

TRUNCATE Statement

- Removes all rows from a table, leaving the table empty and the table structure intact.
- Is a data definition language (DDL) statement rather than a DML statement; cannot easily be undone
- The TRUNCATE statement also efficiently deallocates the space used by the deleted rows which the DELETE statement does not do.
- Additionally, for large tables, TRUNCATE can be faster if there are a number of indexes, triggers etc. on the table.
- `TRUNCATE TABLE` *`table_name`*`;`
- `TRUNCATE TABLE copy_emp;`

Transactional Control Commands (TCL)

- COMMIT—Saves database transactions .
- ROLLBACK—Undoes database transactions .
- SAVEPOINT—Creates points within groups of transactions in which to ROLLBACK
- SET TRANSACTION—Places a name on a transaction

(NOT in scope ?)

DDL – create table , constraints slides to be prepared or not ?

SYNERGETICS GET IT RIGHT | Microsoft Partner Gold Cloud Platform Silver Learning

Thank You