

Instructor Notes:



**Instructor Notes:**

None

## Lesson Objectives



To understand the following topics:

- Joins
  - Oracle Proprietary Joins
  - SQL: 1999 Compliant Joins
- Types of joins
- Sub-queries
- Tips and Tricks

## Instructor Notes:

None

6.1: Joins

## What are Joins?



If we require data from more than one table in the database, then a join is used.

- Tables are joined on columns, which have the same “data type” and “data width” in the tables.
- The JOIN operator specifies how to relate tables in the query.
  - When you join two tables a Cartesian product is formed, by default.
- Oracle supports
  - Oracle Proprietary
  - SQL: 1999 Compliant Joins

**Joins:**

- JOINS make it possible to select data from more than one table by means of a single statement.
- The joining of tables is done in SQL by specifying the tables to be joined in the FROM clause of the SELECT statement.
- When you join two tables a Cartesian product is formed.
- The conditions for selecting rows from the product are determined by the predicates in the WHERE clause.
- All the subsequent WHERE, GROUP BY, HAVING, ORDER BY clauses work on this product.
- If the same table is used more than once in a FROM clause then “aliases” are used to remove conflicts and ambiguities. They are also called as “co-relation names” or “range variables”.

**Instructor Notes:**

- At the moment, do not get into details of all the Types of JOINS.
- Explain each JOIN one by one, as is done in the following slides.

6.1: Joins

**Types of Joins**

Given below is a list of JOINS supported by Oracle:

Oracle Proprietary Joins	SQL: 1999 Compliant Joins
Cartesian Product	Cross Joins
Equijoin	Inner Joins (Natural Joins)
Outer-join	Left, Right, Full outer joins
Non-equijoin	Join on
Self-join	Join Using

**Note:**

- Oracle9i onwards offers JOIN syntax that is SQL: 1999 compliant.
- Prior to the 9i release, the JOIN syntax was different from the ANSI standards.
- The new SQL: 1999 compliant JOIN syntax does not offer any performance benefits over the Oracle proprietary JOIN syntax that existed in prior releases.

**Instructor Notes:**

A Cartesian product is sometimes useful when you want to generate large amount of data for testing purpose

6.1.1: Oracle Proprietary Joins

**Cartesian Joins**

A Cartesian product is a product of all the rows of all the tables in the query. A Cartesian product is formed when the join condition is omitted or it is invalid  
To avoid having Cartesian product always include a valid join condition  
Example

```
SELECT Student_Name, Dept_Name  
FROM Student_Master,  
Department_Master;
```

**Cartesian Product**

Whenever a join condition is completely omitted or is invalid a Cartesian product results. It displays all combinations of rows. A Cartesian product tends generates a large number of rows. Unless there is some specific need to combine all rows avoid a Cartesian product by including a valid join condition in the query

The example shown on the slide joins all rows of Student\_Master and Department Master resulting in a Cartesian Join

**Instructor Notes:**

### Guidelines for Joining Tables



The JOIN condition is written in the WHERE clause

The column names which appear in more than one table should be prefixed with the table name

To improve performance of the query, table name prefix can be include for the other selected columns too

Before we get on to Joins let us understand some basic guidelines to write Join Queries

**Instructor Notes:**

- Explain the use of a decision matrix for simplifying writing JOINS.
- **For example:** If you want to display the name and department number of all the employees who are in the same department as Goyal, you can start by making the following decision tree:

**Columns to Display**

- last\_name column from EMP table
- department\_name column from DEPT table

**Condition**

- last\_name='Goyal'
- employees.department\_id = departments.department\_id
- Now the SQL statement can be easily formulated by looking at the decision matrix. The first column gives the column list in the SELECT statement, the second column gives the tables for the FROM clause, and the third column gives the condition for the WHERE clause.

### 6.1.1: Oracle Proprietary Joins

#### EquiJoin



In an Equijoin, the WHERE statement compares two columns from two tables with the equivalence operator “=”.

This JOIN returns all rows from both tables, where there is a match.

Syntax :

```
SELECT <col1>, <col2>, ...
FROM <table1>, <table2>
Where <table1>.<col1>=<table2>.<col2>
[AND <condition>] [ORDER BY <col1>, <col2>, ...]
```

**Equi Join**

- Equi Join which is sometimes also referred to as Inner Join or simple join is done by writing a join condition using the “=” operator
- Typically the tables are joined to get meaningful data.
- The join is based on the equality of column values in the two tables and therefore is called an Equijoin.
- To join together “n” tables, you need a minimum of “n-1” JOIN conditions.
- **For example:** To join three tables, a minimum of two joins is required.
- In the syntax given in the slide:
  - Column1 in Table1 is usually the Primary key of that table.
  - Column2 in Table2 is a Foreign key in that table.
  - Column1 and Column2 must have the same data type, and for certain data types, they should have same size, as well.

**Instructor Notes:**

- Write the same examples using table aliases in the class and show it to the participants

**6.1.1: Oracle Proprietary Joins**  
**EquiJoin - Example**

Example 1: To display student code and name along with the department name to which they belong

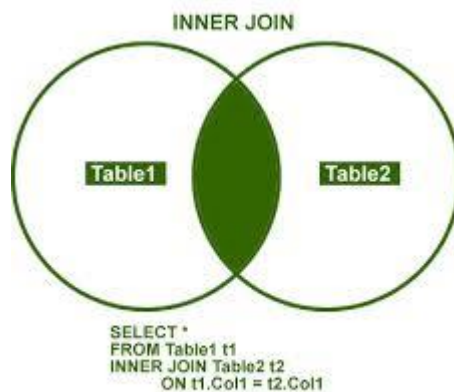
```
SELECT Student_Code, Student_name, Dept_name  
FROM Student_Master , Department_Master  
WHERE Student_Master.Dept_code = Department_Master.Dept_code;
```

Example 2: To display student and staff name along with the department name to which they belong

```
SELECT student_name, staff_name, dept_name  
FROM student_master, department_master, staff_master  
WHERE student_master.dept_code = department_master.dept_code  
and staff_master.dept_code = department_master.dept_code;
```

**Equi Join**

- Frequently, these type of JOIN involves PRIMARY and FOREIGN key complements.
- You can also use table aliases to qualify column names in the SELECT and Join Condition



(C) <http://blog.SQLAuthority.com>



## Instructor Notes:

None

6.1.1: Oracle Proprietary Joins

**Non-EquiJoin**

A non-equi join is based on condition other than an equality operator  
Example: To display details of staff\_members who receive salary in the range defined as per grade

```
SELECT s.staff_name,s.staff_sal,sl.grade
FROM staff_master s,salgrade sl
WHERE staff_sal BETWEEN sl.losal and sl.hisal
```

**Non-Equijoin**

- A Non-equijoin is a JOIN condition containing something other than an equality operator.
- The example on the slide shows a non-equijoin operation

Assume that we have a Salgrade table which is used to determine the range of salary for all staff member. The structure of the table is as follows:

Name	Type
GRADE	NUMBER
LOSAL	NUMBER
HISAL	NUMBER

So to display all the staff members who receive salary between the ranges specified in the salgrade table we will use a non-equijoin

**Instructor Notes:**

None

6.1.1.1: Oracle Proprietary Joins

**Outer Join**

If a row does not satisfy a JOIN condition, then the row will not appear in the query result.

The missing row(s) can be returned by using OUTER JOIN operator in the JOIN condition.

The operator is PLUS sign enclosed in parentheses (+), and is placed on the side of the join(table), which is deficient in information.

**Outer Join**

- If a row does not satisfy the join condition, the row will not appear in the query result. In this situation outer join can be used
- Outer Joins are similar to Inner Joins. However, they give a bit more flexibility when selecting data from related tables. This type of join can be used in situations where it is desired to select "all rows from the table on the left or right", regardless whether they match the join condition
- Outer Join is an exclusive "union" of sets (whereas normal joins are intersection). OUTER JOINS can be simulated using UNIONS.
  - In a JOIN of two tables an Outer Join may be for the first table or the second table. If the Outer Join is taken on, say the DEPARTMENT\_MASTER table, then each row of this table will be selected at least once whether or not a JOIN condition is satisfied.
- An Outer Join does not require each record in the two joint tables to have a matching record in the other table. The joint table retains each record — even if there is no other matching record.

## Instructor Notes:

None

6.1.1: Oracle Proprietary Joins  
Outer Join

## Syntax

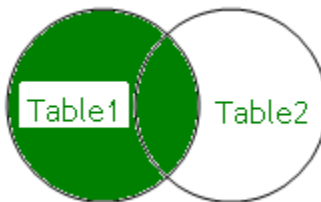
Table1.column = table2.column (+) means OUTER join is taken on table1.  
The (+) sign must be kept on the side of the join that is deficient in information

Depending on the position of the outer join (+), it can be denoted as Left Outer or Right outer Join

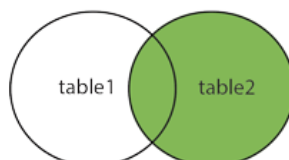
WHERE table1 <OUTER JOIN INDICATOR> = table 2

**Outer Join (contd.):**

- The plus(+) operator can appear only on one side of the expression. It returns those rows from one table that have no direct match in the other table.
- One restriction on outer join is that you cannot use IN operator or the OR operator to create a complex condition
- Left and right outer join diagrams :



## RIGHT OUTER JOIN



**Instructor Notes:**

Tell the participants that if the plus is on the left hand side of equal to operator the right side table will show deficient information and vice versa

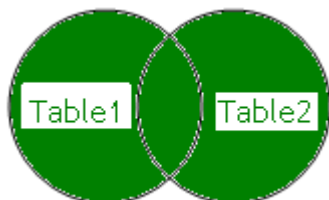
6.1.1: Oracle Proprietary Joins  
**Outer Join - Example**



To display Department details which have staff members and also display department details who do not have any staff members

```
SELECT
staff.staff_code,staff.Dept_Code,dept.Dept_name
      FROM Staff_master staff, Department_Master
dept
      WHERE staff.Dept_Code(+) = dept.Dept_Code
```

Full outer join diagram



## Instructor Notes:

None

### 6.1.1.1: Oracle Proprietary Joins Self Join



In Self Join, two rows from the “same table” combine to form a “resultant row”.

- It is possible to join a table to itself, as if they were two separate tables, by using aliases for table names.
- This allows joining of rows in the same table.

Example: To display staff member information along with their manager information

```
SELECT staff.staff_code, staff.staff_name,  
       mgr.staff_code, mgr.staff_name  
FROM staff_master staff, staff_master mgr  
WHERE staff.mgr_code = mgr.staff_code;
```

**Self Join:**

- Sometimes is required to join the table to itself. To join a table to itself, “two copies” of the same table have to be opened in the memory.
- Since the table names are the same, the second table will overwrite the first table. In effect, this will result in only one table being in memory.
  - This is because a table name is translated into a specific memory location.
- Hence in the FROM clause, the table name needs to be mentioned twice with an “alias”
  - These two table aliases will cause two identical tables to be opened in different memory locations.
  - This will result in two identical tables to be physically present in the computer memory.

**Instructor Notes:**

None

6.2: Subqueries

**What is a SubQuery?**

A sub-query is a form of an SQL statement that appears inside another SQL statement.

- It is also called as a “nested query”.

The statement, which contains the sub-query, is called the “parent statement”.

The “parent statement” uses the rows returned by the sub-query.

**Sub-queries:**

- As mentioned earlier, since a basic SQL query returns a relation, it can be used to construct composite queries.
- Such a SQL query, which is nested within another higher level query, is called a “sub-query”.
- This kind of a nested sub-query is useful in cases, where we need to select rows from tables based on a condition, which depends on the data stored in the table itself.
- These can be useful when you need to select rows from a table with a condition that depends on data within the table itself.

## Instructor Notes:

None

6.2: Subqueries

## Subquery - Examples



Example 1: To display name of students from "Mechanics" department.

- Method 1:

```
SELECT Dept_Code  
FROM Department_Master  
WHERE Dept_name = 'Mechanics';
```

- O/P : 40

```
SELECT student_code,student_name  
FROM student_master  
WHERE dept_code=40;
```

Consider the example given on the slide. We want to find details of students from "Mechanics" department. As you can see two queries are used to resolve these problem. Instead of that you can resolve this problem by combining both the queries into one as shown on the next slide.

## Instructor Notes:

6.2: Subqueries

## Subquery - Examples



Example 1 (contd.):

- Method 2: Using sub-query

```
SELECT student_code, student_name
  FROM student_master
 WHERE dept_code = (SELECT dept_code
                    FROM department_master
                    WHERE dept_name =
'Mechanics');
```

The problem is resolved using a one query inside another. The inner query is called as a subquery or nested query. The subquery result is used by the outer query. The subqueries can have more levels of nesting. The innermost query will execute first. The subquery execute only once before the outer query



**Instructor Notes:**

Tell the participants that in this lesson we will only see subqueries in Select statement clauses. Others we will cover later like creating views, creating tables etc...

6.2: Subqueries

**Where to use Subqueries?**

Subqueries can be used for the following purpose :

- To insert records in a target table.
- To create tables and insert records in the table created.
- To update records in the target table.
- To create views.
- To provide values for conditions in the clauses, like WHERE, HAVING, IN, etc., which are used with SELECT, UPDATE and DELETE statements.

- When the WHERE clause needs a set of values which can be only obtained from another query, the Sub-query is used. In the WHERE clause it can become a part of the following predicates
  - COMPARISON Predicate
  - IN Predicate
  - ANY or ALL Predicate
  - EXISTS Predicate.
- It can be also used as a part of the condition in the HAVING clause.

## Instructor Notes:

None

## 6.2: Subqueries

## Comparison Operators for Subqueries



## Types of SubQueries

- Single Row Subquery
- Multiple Row Subquery.

Some comparison operators for subqueries:

Operator	Description
IN	Equals to any member of
NOT IN	Not equal to any member of
*ANY	compare value to every value returned by sub-query using operator *
*ALL	compare value to all values returned by sub-query using operator *

**Sub-queries by using Comparison operators:**

- Sub-queries are divided as “single row” and “multiple row” sub-queries. While single row comparison operators (=, >, >=, <, <=, <>) can only be used in single row sub-queries, multiple row sub-queries can use IN, ANY or ALL.
- **For example:** The assignment operator (=) compares a single value to another single value. In case a value needs to be compared to a list of values, then the IN predicate is used. The IN predicate helps reduce the need to use multiple OR conditions.
- The NOT IN predicate is the opposite of the IN predicate. This will select all rows where values do not match the values in the list.
- The FOR ALL predicate is evaluated as follows:
  1. True if the comparison is true for every value of the list of values.
  2. True if sub-query gives a null set (No values)
  3. False if the comparison is false for one or more of the list of values generated by the sub-query.
- The FOR ANY predicate is evaluated as follows
  1. True if the comparison is true for one or more values generated by the sub-query.
  2. False if sub-query gives a null set (No values).
  3. False if the comparison is false for every value of the list of values generated by the sub-query.

## Instructor Notes:

None

## 6.2: Subqueries

## Using Comparison Operators - Examples



Example 1: To display all staff details of who earn salary least salary

```
SELECT staff_name, staff_code, staff_sal
FROM staff_master
WHERE staff_sal = (SELECT MIN(staff_sal)
                  FROM staff_master) ;
```

Example 2: To display staff details who earn salary greater than average salary earned in dept 10

```
SELECT staff_code, staff_sal
FROM staff_master
WHERE staff_sal > ANY(SELECT AVG(staff_sal)
                     FROM staff_master GROUP BY
dept_code);
```

- For Single Row Subquery the sub-query should result in one value of the same data type as the left-hand side.
- Similarly for Multiple Row Subquery the list of values generated by the sub-query should be of same data type as the left-hand side
- The slide above shows examples of Subqueries. The first query is an example of Single Row Subquery and the second is an example of Multiple row Subquery

**Instructor Notes:**

6.3: Tips and Tricks

**Quick Guidelines****For Using Subqueries**

- Should be enclosed in parenthesis
- They should be placed on the right side of the comparison condition
- Use operator carefully. Single Row operators for Single Row Subquery and Multiple Row operator for Multiple Row Subquery

**Instructor Notes:**

6.3: Tips and Tricks

**Quick Guidelines**

If Single row operators are used for a sub query that returns multiple rows, Oracle would throw an error

Restrict using the NOT IN clause, which offers poor performance because the optimizer has to use a nested table scan to perform this activity.

**Instructor Notes:**

None

### Summary



In this lesson, you have learnt:

- Joins
- Oracle Proprietary Joins
- Sub-queries



**Instructor Notes:****Answers for Match the Following:**

- 1 - b
- 2 - a
- 3 - d
- 4 - c

**Review – Match the Following**

1. Equi Join

a. is based on any other operator other than equality

2. Non-equijoin

b. Is based on equality operator

3. Outer Join

c. Joins the table to itself

4. Self Join

d. includes a "+" operator with equality operator



**Instructor Notes:****Answers for Review Questions:****Question 1:**

Answer: Option 2.

**Question 2:**

Answer: Nested query

**Review – Questions**

Question 1: The SQL compliant join which is same as EquiJoin.

- Option 1: Cross Join
- Option 2: Natural Join
- Option 3: Full Outer Join

Question 2: A sub-query is also sometimes termed as \_\_\_\_\_.





**Instructor Notes:****Answers for Review Questions:****Question 3:**

Answer: True

**Question 4:**

Answer: Option 2, and Option 4

**Review – Questions**

Question 3: A sub-query can be used for creating and inserting records.

- True / False

Question 4: If a sub-query returns multiple values, then the valid operators is/are \_\_\_\_.

- Option 1: =
- Option 2: IN
- Option 3: >
- Option 4: Any

