

# DBMS SQL

## Lesson 6: Joins and Subqueries

## Lesson Objectives

- To understand the following topics:
  - Join
    - Oracle Proprietary Joins
    - SQL: 1999 Compliant Joins
  - Sub-queries
    - Co-related sub-query
    - Exists / Non-Exists Operators
  - CONNECT BY and START WITH clauses



## 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



Copyright © Capgemini 2015. All Rights Reserved. 3

### 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”.

**Joins (contd.):**

Assume two tables:

**TABLE F1**

| <u>COL1</u> | <u>COL2</u> |
|-------------|-------------|
| A           | 1           |
| B           | 2           |
| C           | 3           |

**Table F2**

| <u>COL1</u> | <u>COL2</u> |
|-------------|-------------|
| X           | 100         |
| Y           | 200         |

The statement `SELECT * FROM F1,F2;` results in:

| <u>COL1</u> | <u>COL2</u> | <u>COL1</u> | <u>COL2</u> |
|-------------|-------------|-------------|-------------|
| A           | 1           | X           | 100         |
| B           | 2           | X           | 100         |
| C           | 3           | X           | 100         |
| A           | 1           | Y           | 200         |
| B           | 2           | Y           | 200         |
| C           | 3           | Y           | 200         |

6 rows selected

The statement `SELECT * FROM F1 First_T, F1 Second_T;` results in:

| <u>COL1</u> | <u>COL2</u> | <u>COL1</u> | <u>COL2</u> |
|-------------|-------------|-------------|-------------|
| A           | 1           | A           | 1           |
| B           | 2           | A           | 1           |
| C           | 3           | A           | 1           |
| A           | 1           | B           | 2           |
| B           | 2           | B           | 2           |
| C           | 3           | B           | 2           |
| A           | 1           | C           | 3           |
| B           | 2           | C           | 3           |
| C           | 3           | C           | 3           |

9 rows selected

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 on                       |

### 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.

As we go ahead we will see both variations of joins supported by Oracle.

6.2: 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;
```



Copyright © Capgemini 2015. All Rights Reserved. 8

### 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 to generate 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.

## 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

6.2: 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>, ...]
```

Copyright © Capgemini 2015. All Rights Reserved. 8

### 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 the same size, as well.



6.2: 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;
```



Copyright © Capgemini 2015. All Rights Reserved. 9

### 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

6.2: 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
```



Copyright © Capgemini 2015. All Rights Reserved. 10

### 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

6.2: 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.

WHERE table1 <OUTER JOIN INDICATOR> = table 2



Copyright © Capgemini 2015. All Rights Reserved. 11

### 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.

## 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

### 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

## 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
```

6.2: 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;
```



Copyright © Capgemini 2015. All Rights Reserved. 58

### Self Join:

Sometimes it 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.

## 6.3: SQL:1999 Compliant Joins

## SQL: 1999 Compliant Joins - Syntax

- Syntax:

```
SELECT table1.column, table2.column  
FROM table1  
[CROSS JOIN table2] |  
[NATURAL JOIN table2] |  
[JOIN table2 USING (column_name)] |  
[JOIN table2 ON (table1.column_name =  
                 table2.column_name)] |  
[LEFT|RIGHT|FULL OUTER JOIN table2  
 ON (table1.column_name = table2.column_name)];
```

### SQL:1999 Compliant Joins

The SQL Compliant joins can obtain the similar results that we have discussed in the previous slides. They differ only in syntax.

The SQL compliant joins are supported from Oracle 9i version onwards

6.3: SQL:1999 Compliant Joins

## Cross Join

- The Cross Join and Cartesian product are same which produces the cross-product of the tables

Example: Cross Join on Student\_Master and Department\_Master

```
SELECT student_name, dept_name  
FROM student_master  
CROSS JOIN department_master;
```



Copyright © Capgemini 2015. All Rights Reserved. 15

### Cross Join

The example on the slide create a cross product of the two tables.  
The query result is same as the following query:

```
SELECT student_name,dept_name  
FROM Student_Master, Department_Master;
```



## 6.3: SQL:1999 Compliant Joins

## Natural Join

- The Natural Join is based on the all columns that have same name and datatype in the tables include in the query
- All the rows that have equal values in the matched columns are fetched

Example: To display student details along with their department details

```
SELECT Student_Code, Student_name, Dept_Code,  
       Dept_name  
FROM Student_Master  
NATURAL JOIN Department_Master
```



Copyright © Capgemini 2015. All Rights Reserved. 17

### Natural Join

Prior to Oracle 9i, without explicitly specifying the columns of the corresponding tables a join was not possible. With the Natural Join clause, the join can happen automatically based on column names that match in name and datatype.

Oracle returns an error if the datatypes of the columns are different though they have the same name.

The Natural Join is same as EquiJoin.

6.3: SQL:1999 Compliant Joins

## USING clause

- The USING clause can be replace the NATURAL JOIN if the columns have same names but data types do not match.
- The table name or aliases should not be used in the referenced columns
- This clause should be used to match only one column when there are more than one column matches

### USING clause

The NATURAL JOIN returns an error if the datatypes of matching columns are different. In that case the USING clause can be used to specify only those columns on which the join has to done.

The NATURAL JOIN and USING clause are mutually exclusive.

The column used in USING clause cannot be used in WHERE clause

## USING clause - Example

Example 1: To display student details along with their department details. The department code does not match in datatype, hence the join is performed with the USING clause

```
SELECT student_code, student_name, dept_code, dept_name  
FROM student_master  
JOIN department_master  
USING (dept_code, dept_code);
```

6.3: SQL:1999 Compliant Joins

## ON clause

- Explicit join condition can be specified by using ON clause
- Other search conditions can be specified in addition to join condition

Example: To display student along with department details from Computer Science department

```
SELECT student.student_code, student.student_name,  
       student.dept_code, dept.dept_name  
FROM student_master student  
JOIN department_master dept  
ON (student.dept_Code = dept.dept_Code)  
AND dept.dept_Name ='Computer Science' ;
```



Copyright © Capgemini 2015. All Rights Reserved. 30

### ON clause

With the ON clause you can specify join conditions separate from any other search conditions.

The query is readable and easy to understand

6.3: SQL:1999 Compliant Joins

## LEFT, RIGHT & FULL Outer Join

- 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

### LEFT, RIGHT and FULL OUTER JOIN

The SQL compliant outer join is similar to Oracle proprietary outer join except for the fact that it also has support to perform FULL OUTER JOIN

## LEFT, RIGHT & FULL Outer Join - Example

Example 1: Display student & department details and also those departments who do not have students

```
SELECT s.student_code, s.dept_code, d.dept_name  
FROM student_master s  
RIGHT OUTER JOIN department_master d  
ON (s.dept_code = d.dept_code);
```

Example 2 Display student & department details, also those students who are not assigned to any department

```
SELECT s.student_code, s.dept_code, d.dept_name  
FROM student_master s  
LEFT OUTER JOIN department_master d  
ON (s.dept_code = d.dept_code);
```

## LEFT, RIGHT & FULL Outer Join - Example

Example 3: Display student & department details. Also those departments who do have students and students who are not assigned to any department

```
SELECT s.student_code,s.dept_code,d.dept_name  
FROM student_master s  
FULL OUTER JOIN department_master d  
ON (s.dept_code = d.dept_code );
```

## 6.4: 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.



## 6.4: 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;
```



Copyright © Capgemini 2015. All Rights Reserved. 25

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.

## 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

## 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.

## 6.3: 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:

True if the comparison is true for every value of the list of values.

True if sub-query gives a null set (No values)

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

True if the comparison is true for one or more values generated by the sub-query.

False if sub-query gives a null set (No values).

False if the comparison is false for every value of the list of values generated by the sub-query.

## 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 WHERE dept_code=10);
```

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

## 6.4: Co-related Subquery

## What is a Co-related Subquery?

- A sub-query becomes “co-related”, when the sub-query references a column from a table in the “parent query”.
- A co-related sub-query is evaluated once for each row processed by the “parent statement”, which can be either SELECT, UPDATE, or DELETE statement.
- A co-related sub-query is used whenever a sub-query must return a “different result” for each “candidate row” considered by the “parent query”.

### Co-related Sub-query:

A “co-related sub-query” is a form of query used in SELECT, UPDATE, or DELETE commands to force the DBMS to evaluate the query once “per row” of the parent query, rather than once for the “entire query”.

A co-related sub-query is used to answer “multi-part questions”, whose answers depend on the values in each row of the parent query.

A co-related sub-query is one way of reading every row in a table, and comparing values in each row against related data.

## Co-related Subquery -Examples

- Example 2: To display staff details whose salary is greater than the average salary in their own department:

```
SELECT staff_name, staff_sal , dept_code  
FROM staff_Master s  
WHERE staff_sal > (SELECT AVG(staff_sal)  
FROM staff_Master m  
WHERE s.dept_code = m.dept_code );
```

## 6.4: Co-related Subquery

## EXISTS/ NOT EXISTS Operator

- The EXISTS / NOT EXISTS operator enables to test whether a value retrieved by the Outer query exists in the result-set of the values retrieved by the Inner query.
- The EXISTS / NOT EXISTS operator is usually used with a co-related sub-query.
  - If the query returns at least one row, the operator returns TRUE.
  - If the value does not exist, it returns FALSE.
- The NOT EXISTS operator enables to test whether a value retrieved by the Outer query is not a part of the result-set of the values retrieved by the Inner query.

The Exists Predicate is of the form

<Query> WHERE  
EXISTS < sub-query>

Query will be evaluated if EXISTS takes a value TRUE. It takes the Value TRUE if the <sub-query> result table is non null and FALSE if it is Null. It is mostly used with Co-Related Subqueries. EXISTS does not check for a particular value. It checks whether subquery returns rows or not.



## EXISTS/ NOT EXISTS Operator - Examples

- Example 1: To display details of employees who have some other employees reporting to them.

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

- Example 2: To display details of departments which have employees working in it.

```
SELECT dept_code,dept_name FROM department_master
WHERE EXISTS ( SELECT dept_code FROM staff_master
WHERE staff_master.dept_code =
department_master.dept_code) ;
```

## CONNECT BY and START WITH Clauses

- The START WITH .. CONNECT BY clause can be used to select data that has a hierarchical relationship
  - Usually, they have some sort of parent-child relationship.
  - They are used to retrieve rows, which are connected to each other through a tree-like structure.

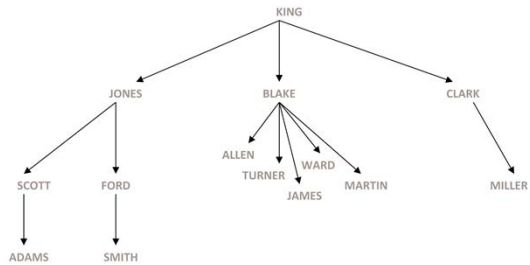
### CONNECT BY and START WITH Clauses:

A pseudocolumn behaves like a table column, but is not actually stored in the table. You can select from pseudocolumns, but you cannot insert, update, or delete their values.

For each row returned by a hierarchical query, the LEVEL pseudocolumn returns 1 for a root row, 2 for a child of a root, and so on. A root row is the highest row within an inverted tree. A child row is any nonroot row. A parent row is any row that has children. A leaf row is any row without children

## CONNECT BY and START WITH Clauses

- The earliest ancestor in the tree is called the root-node called as a trunk. Extending from the trunk are branches, which have other branches.



## CONNECT BY and START WITH Clauses

- The restrictions on SELECT statements performing hierarchical queries are as follows :
  - A SELECT statement that performs a hierarchical query cannot perform a JOIN.
  - If an ORDER BY clause is used in a hierarchical query, then Oracle orders rows using the ORDER BY clause rather than in a hierarchical fashion.

## CONNECT BY, START WITH Clauses-Examples

- Example 1: To list “Allen” and his subordinates

```
SELECT staff_name, staff_code, mgr_code  
FROM staff_master  
CONNECT BY PRIOR staff_code = mgr_code  
START WITH staff_name = 'Allen';
```

Note: If START WITH clause is omitted, then the tree structure is generated for each of the rows in the EMP table.

6.5: 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
- Cannot use ORDER By clause in subquery unless performing top-n analysis
- Use operator carefully. Single Row operators for Single Row Subquery and Multiple Row operator for Multiple Row Subquery



## Quick Guidelines

- Restrict using the NOT IN clause, which offers poor performance because the optimizer has to use a nested table scan to perform this activity.
- Instead try to use one of the following options, all of which offer better performance:
  - Use EXISTS or NOT EXISTS
  - Use IN
  - Perform a LEFT OUTER JOIN and check for a NULL condition



## Quick Guidelines

- If you have a choice of using the IN or the EXISTS clauses in your SQL, use the EXISTS clause as it is usually more efficient and performs faster.
  - Consider EXISTS in place of table joins.
  - Consider NOT EXISTS in place of NOT IN.





## Summary

- In this lesson, you have learnt:
  - Joins
  - Oracle Proprietary Joins
  - SQL: 1999 Compliant Joins
- Sub-queries
  - Co-related sub-query
  - Exists / Non-Exists Operators
- CONNECT BY and START WITH clauses



## Review – Match the Following

1. Equi Join

a. is based on any other operator other than equality

2. Non-equi join

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



## 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 \_\_\_\_.



## 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

