DBMS SQL

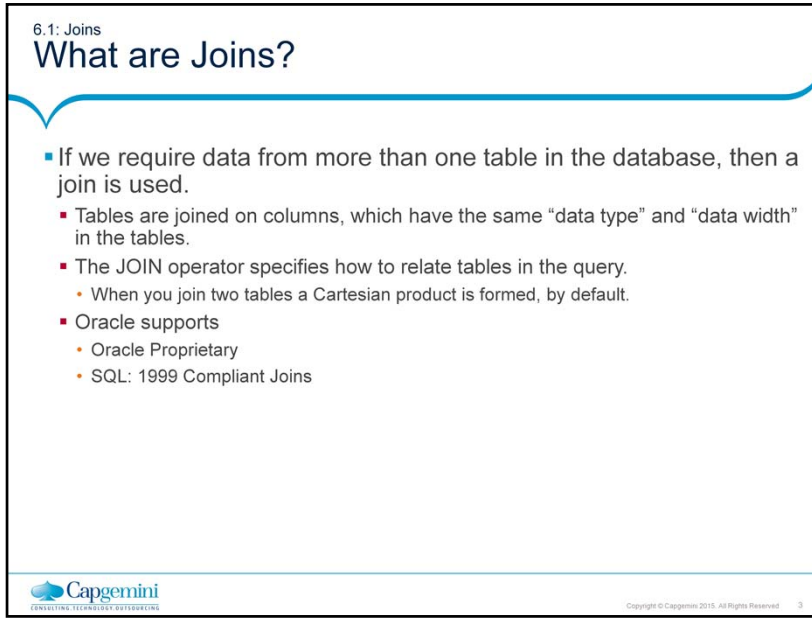
Lesson 6: Joins and Subqueries

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

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



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 ambiquities. They are also called as “co-relation names” or “range variables”.

Joins (contd.):
Assume two tables:
TABLE F1

COL1	COL2
A	1
B	2
C	3

Table F2

COL1	COL2
X	100
Y	200

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

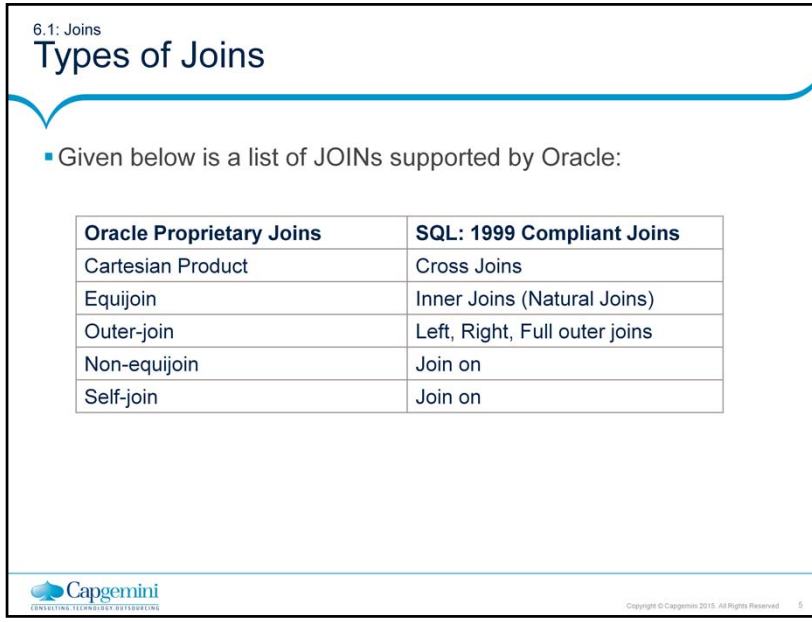
COL1	COL2	COL1	COL2
A	1	X	100
B	2	X	100
C	3	X	100
A	1	Y	200
B	2	Y	200
C	3	Y	200

6 rows selected

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

COL1	COL2	COL1	COL2
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



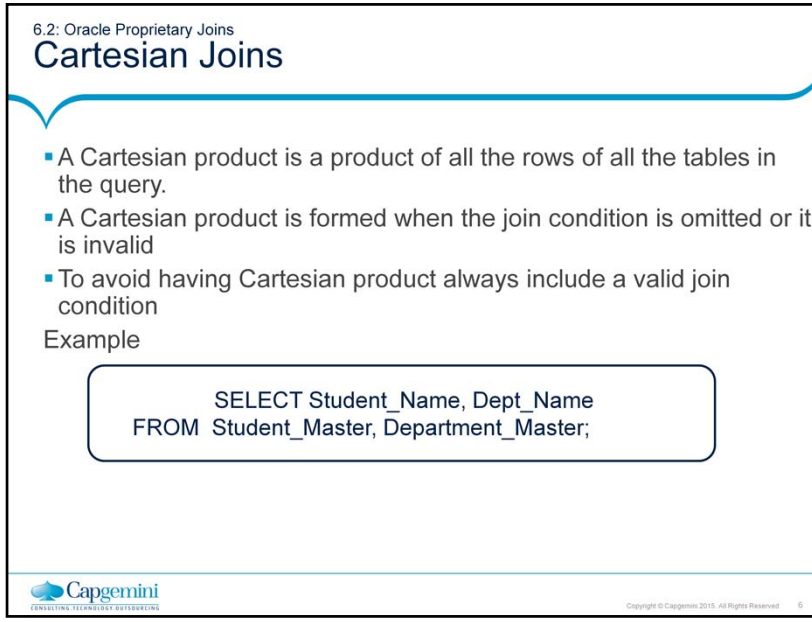
Note:

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.

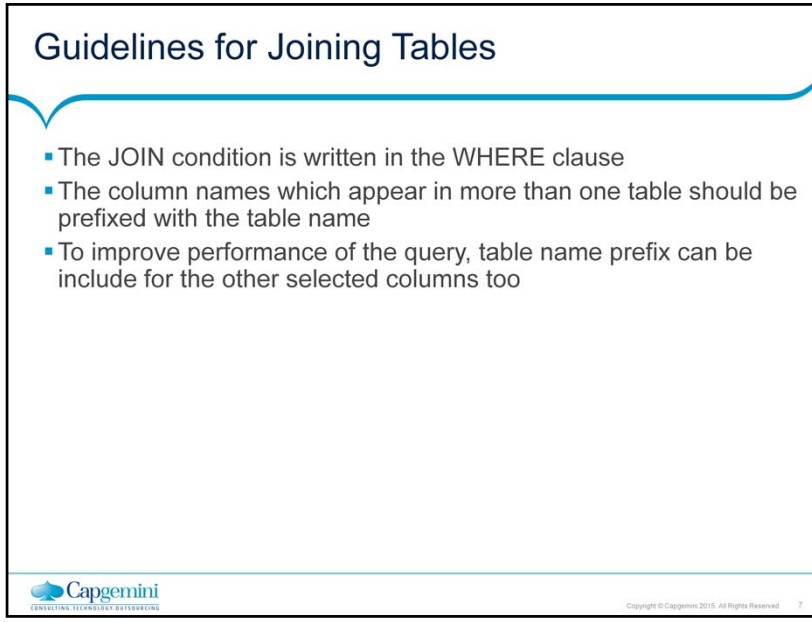
As we go ahead we will see both variations of joins supported by Oracle.



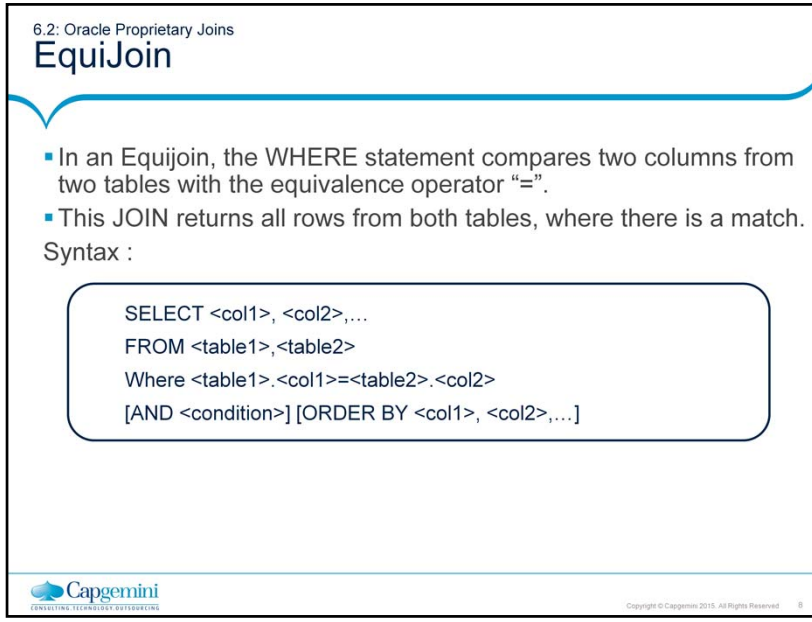
Cartesian Product

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



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



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 joint 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.

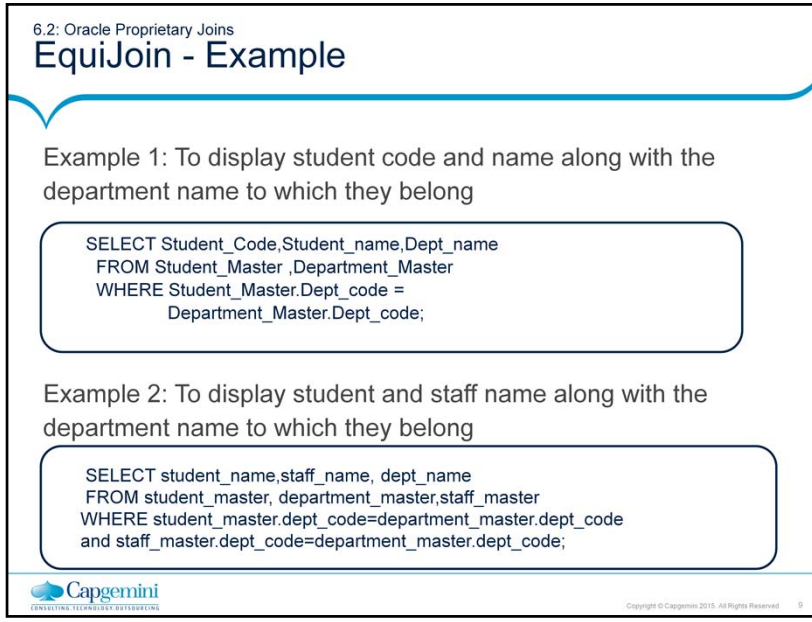
For example: To join three tables, a minimum of two joins is required.

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.

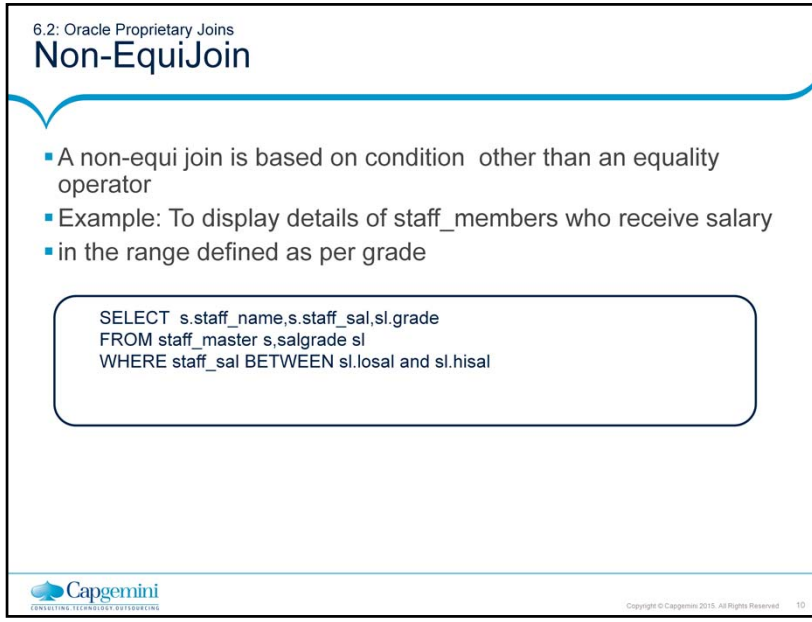
Column1 and Column2 must have the same data type, and for certain data types, they should have same size, as well.



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



Non-Equijoin

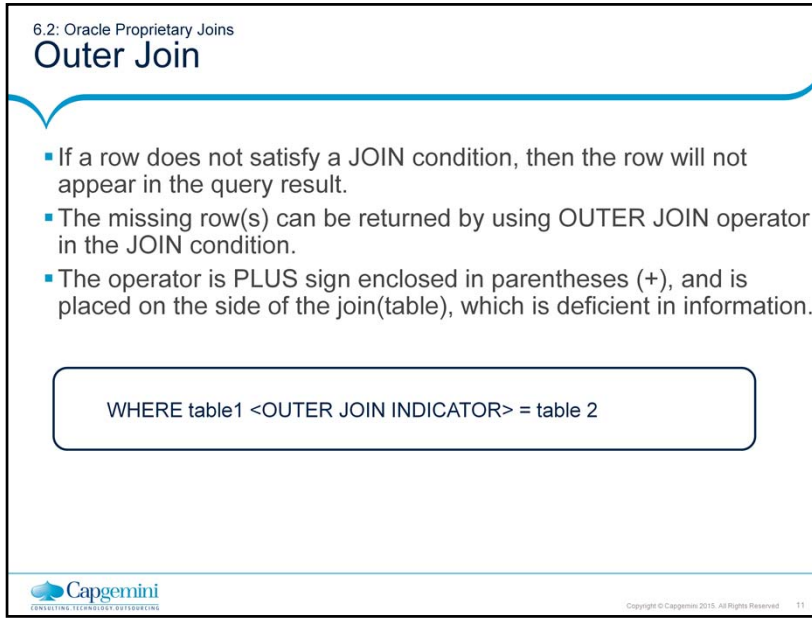
A Non-equijoin is a JOIN condition containing something other than an equality operator.

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



Outer Join

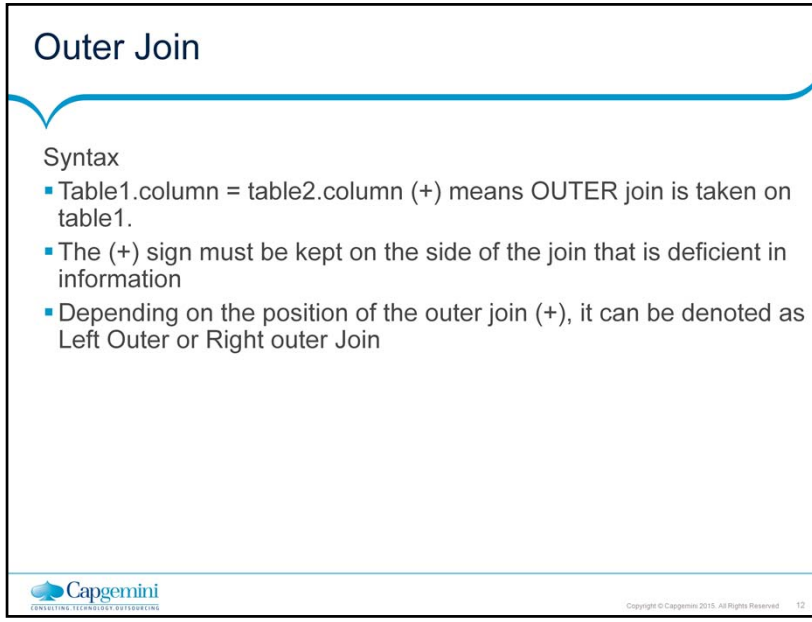
If a row does not satisfy the join condition, the row will not appear in the query result. In this situation outer join can be used

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 (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

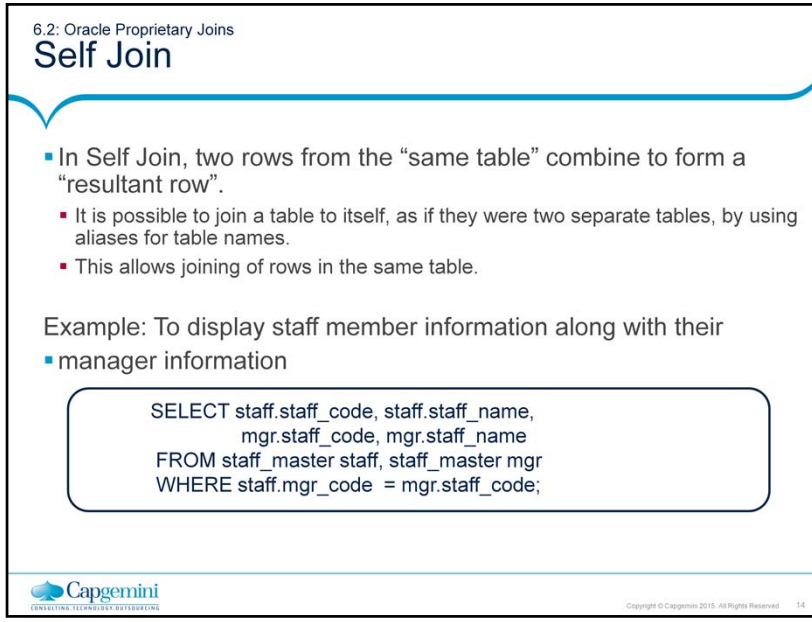
Outer Join - Example

- 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
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 13



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.

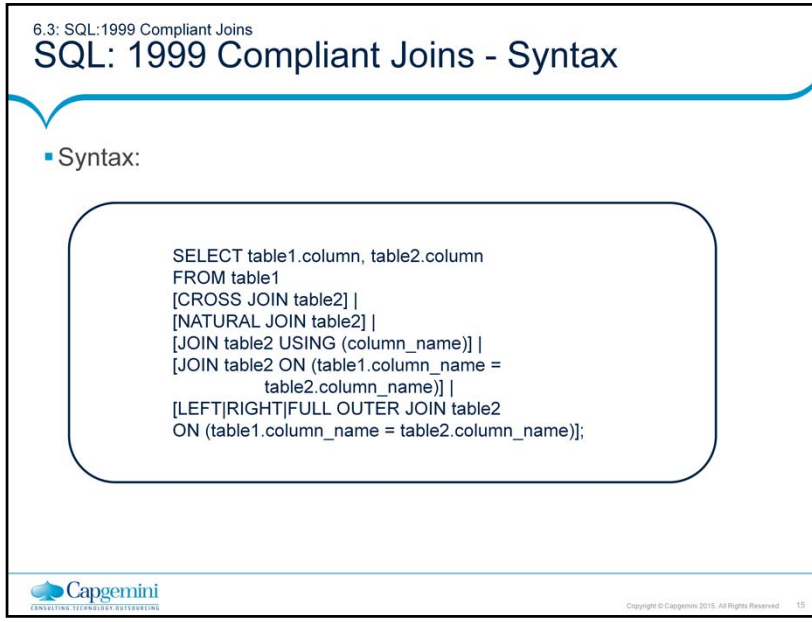
Since the table names are the same, the second table will overwrite the first table. In effect, this will result in only one table being in memory.

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.

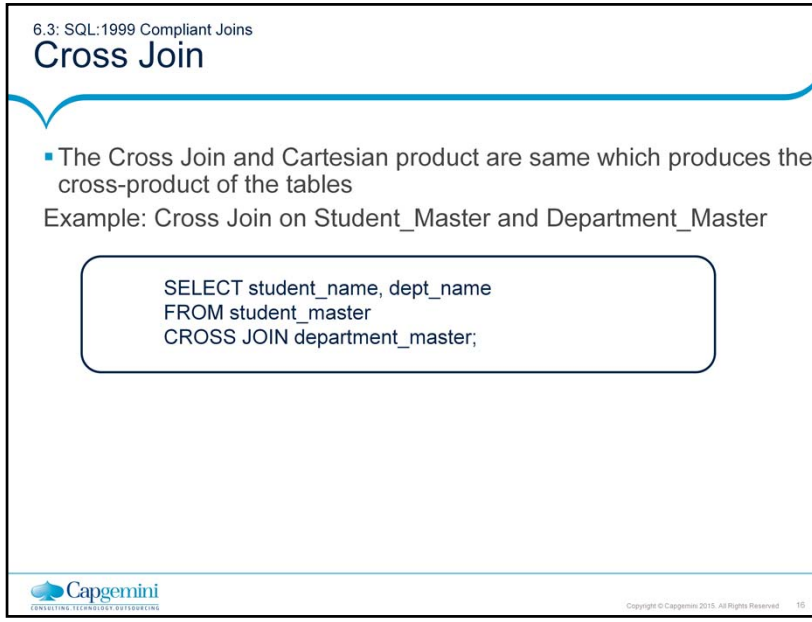
This will result in two identical tables to be physically present in the computer memory.



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.

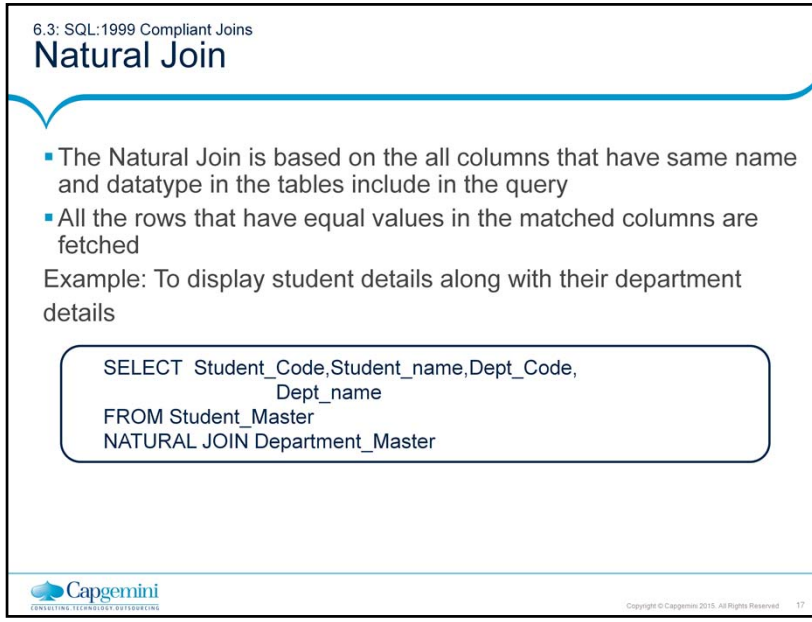
The SQL compliant joins are supported from Oracle 9i version onwards



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;
```

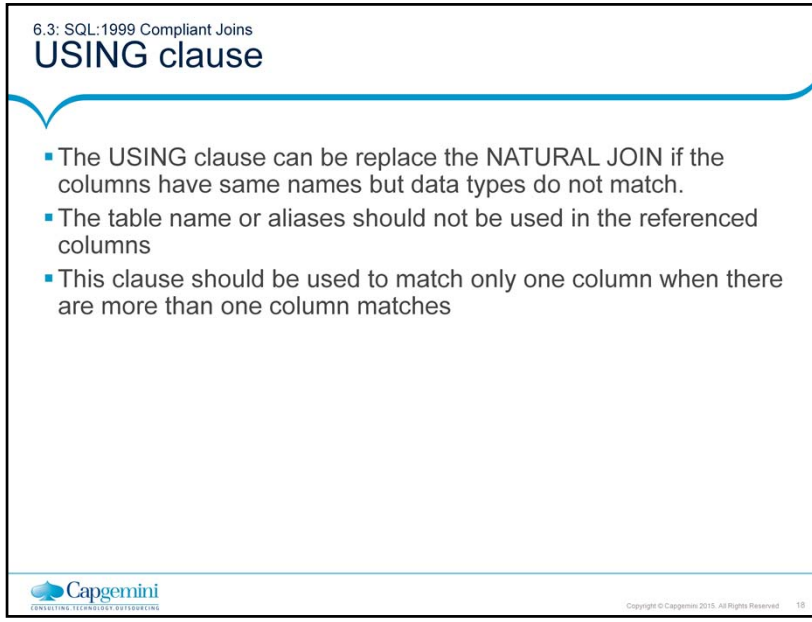


Natural Join

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.



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

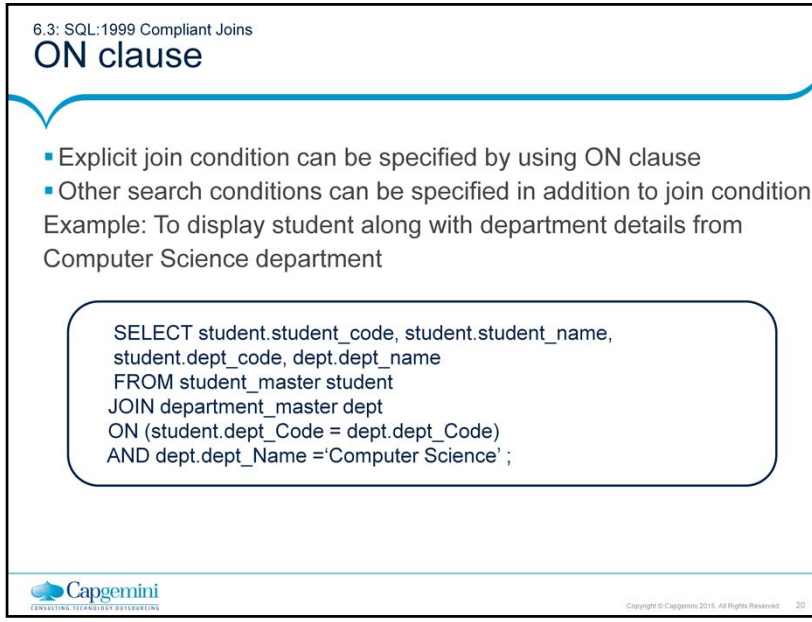
USING clause - Example

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);
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

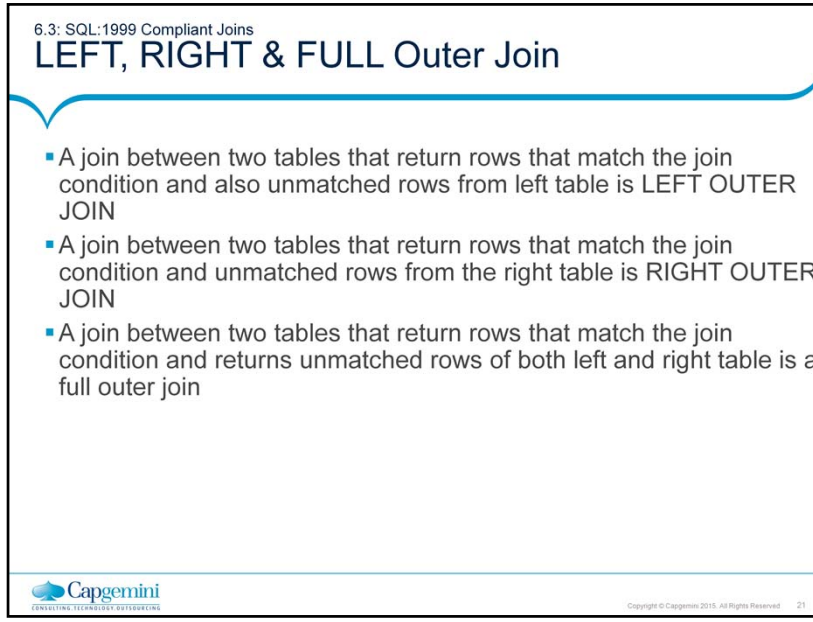
 19



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



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 thos departments who do 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);
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 22

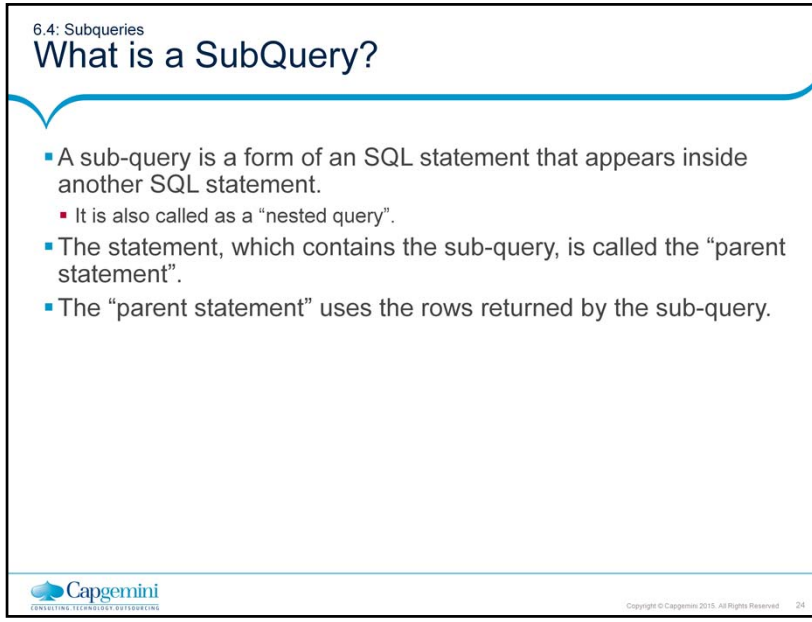
LEFT, RIGHT & FULL Outer Join - Example

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 );
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 23



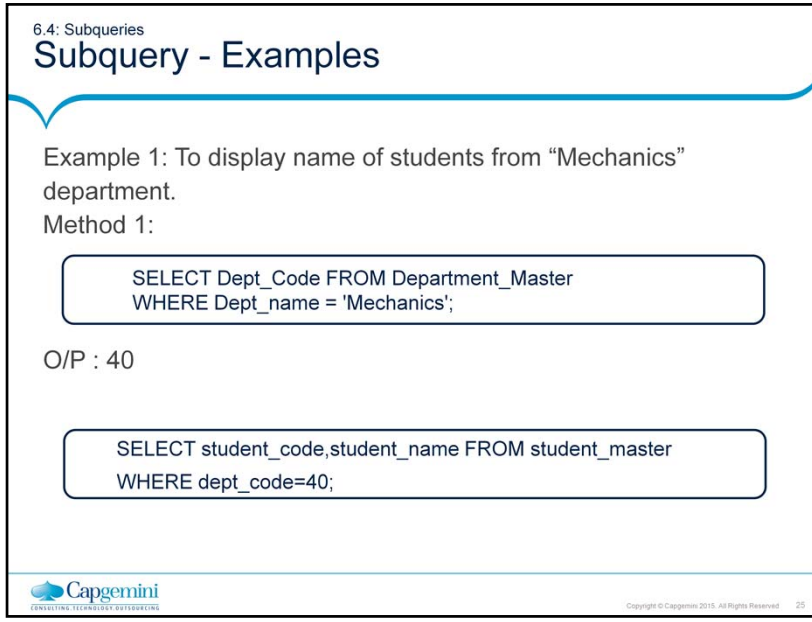
Sub-queries:

As mentioned earlier, since a basic SQL query returns a relation, it can be used to construct composite queries.

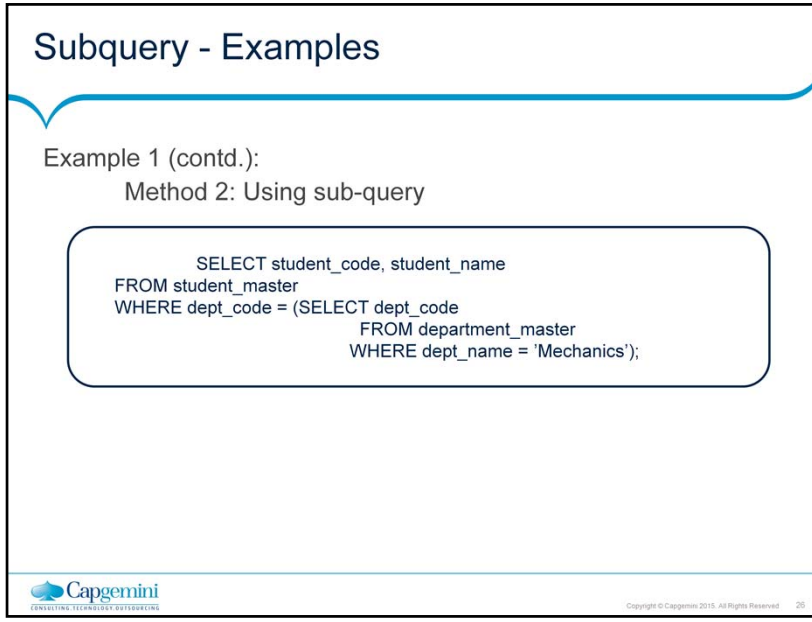
Such a SQL query, which is nested within another higher level query, is called a “sub-query”.

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.

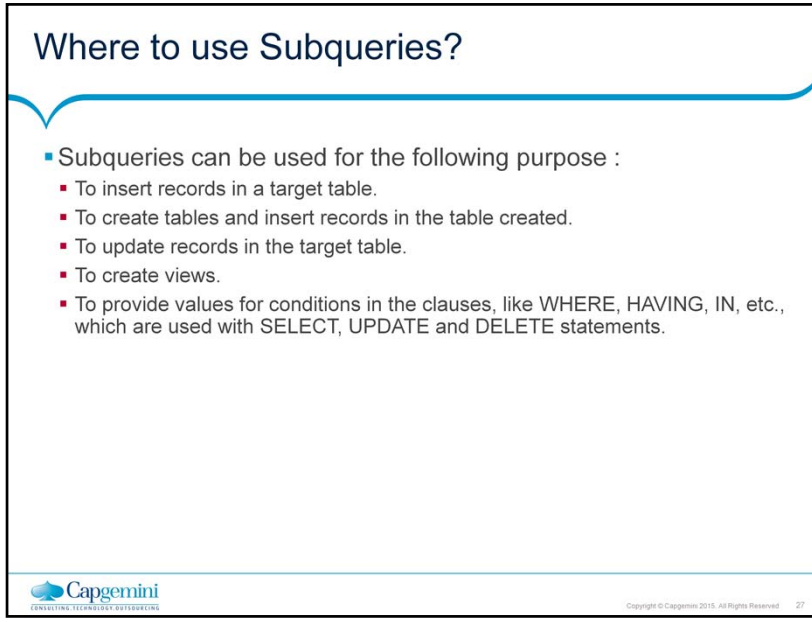
These can be useful when you need to select rows from a table with a condition that depends on data within the table itself.



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.



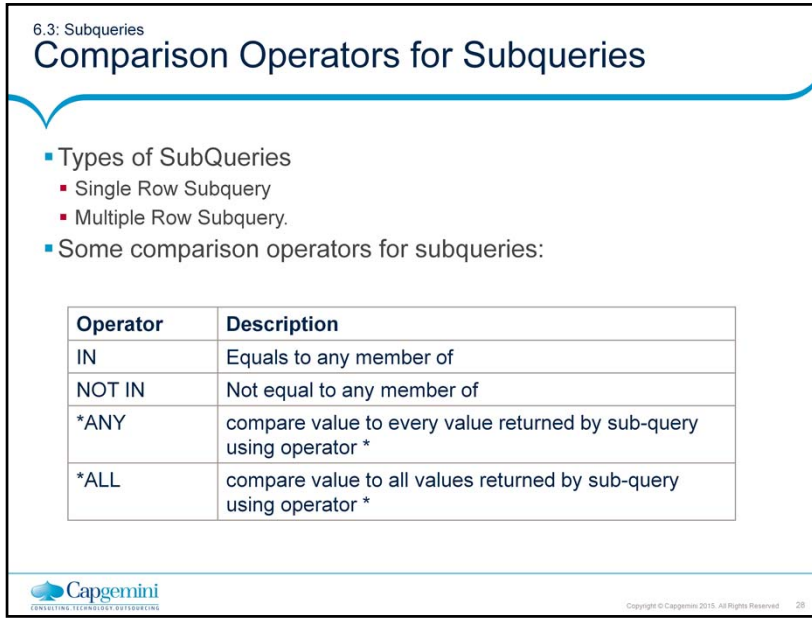
The problem is resolved using a one query inside another. The inner query is called as a subquery or nested query. The subquery result is used by the outer query. The subqueries can have more levels of nesting. The innermost query will execute first. The subquery execute only once before the outer query



• 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.



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)

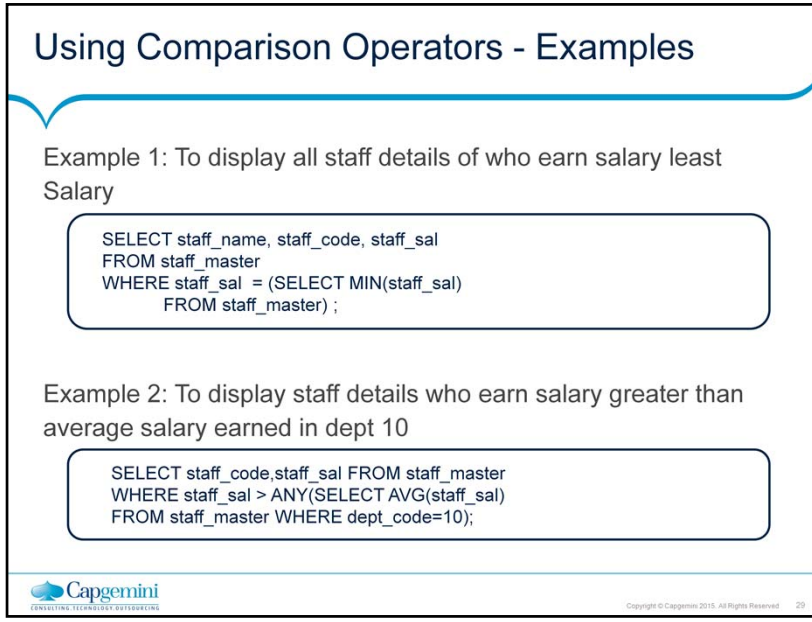
False if the comparison is false for one or more of the list of values generated by the sub-query.

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).

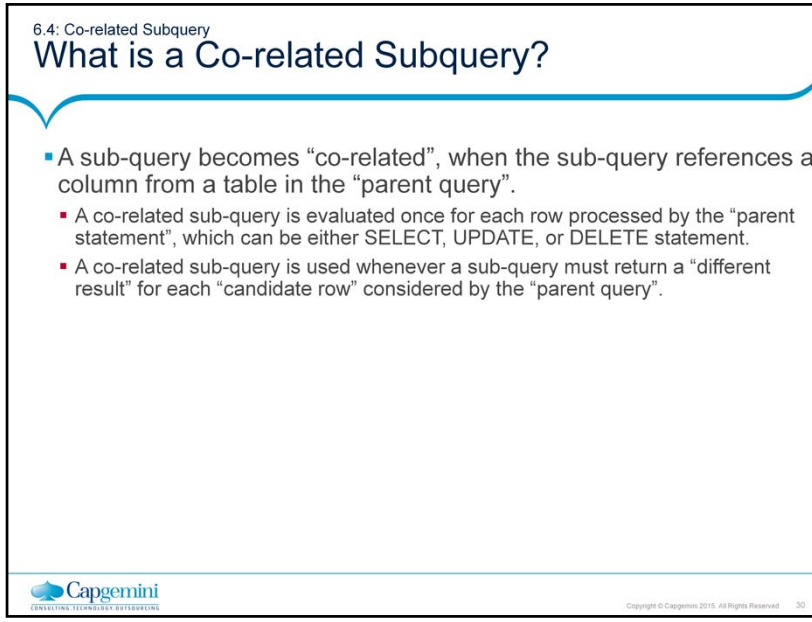
False if the comparison is false for every value of the list of values generated by the sub-query.



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



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.

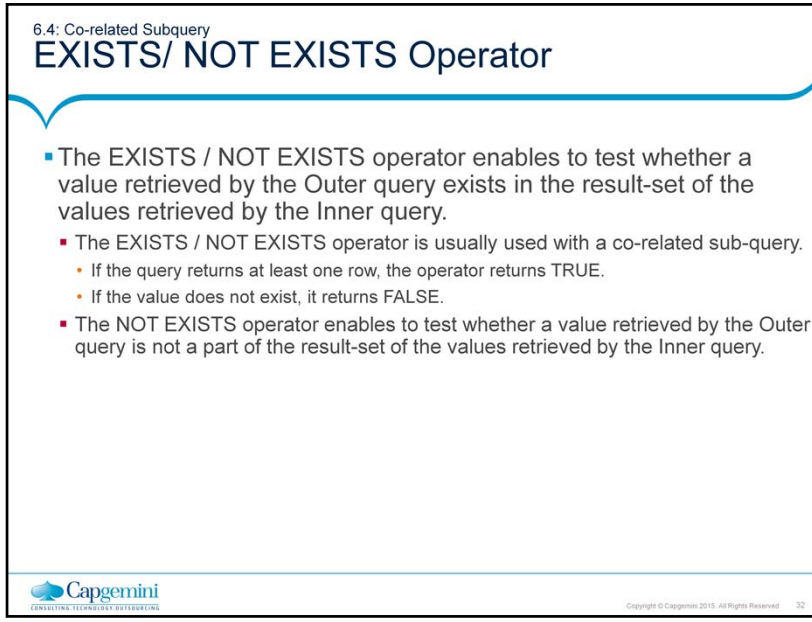
Co-related Subquery -Examples

- 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 );
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 31



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.

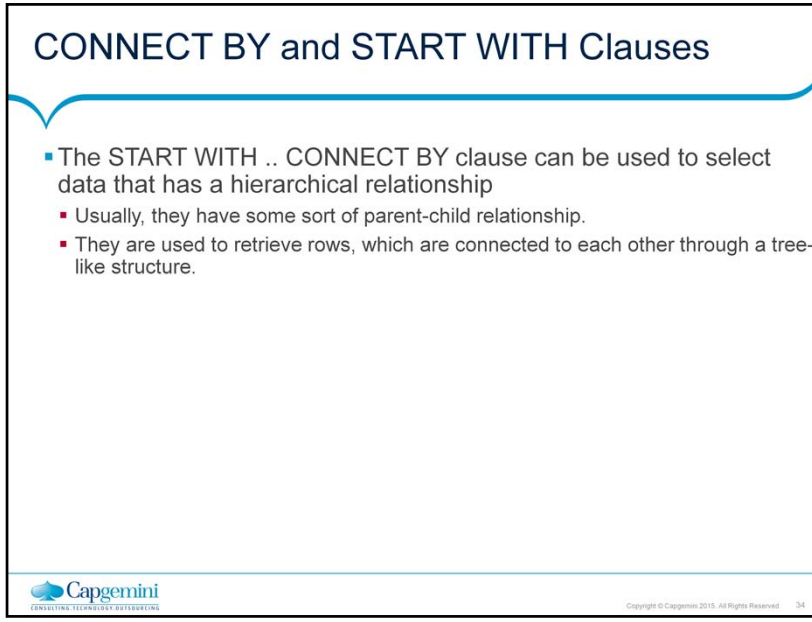
```
SELECT staff_code, staff_name FROM staff_master staff
WHERE  EXISTS (SELECT mgr_code FROM staff_master mgr WHERE
mgr.mgr_code = staff.staff_code) ;
```

- 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) ;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 33



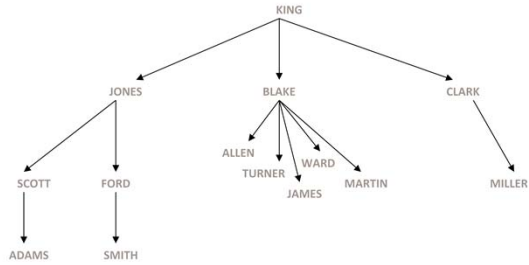
CONNECT BY and START WITH Clauses:

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.



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 35

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.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 36

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

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 38

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

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

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.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 40

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

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 41

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

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 42

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 ___.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 43

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

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 44