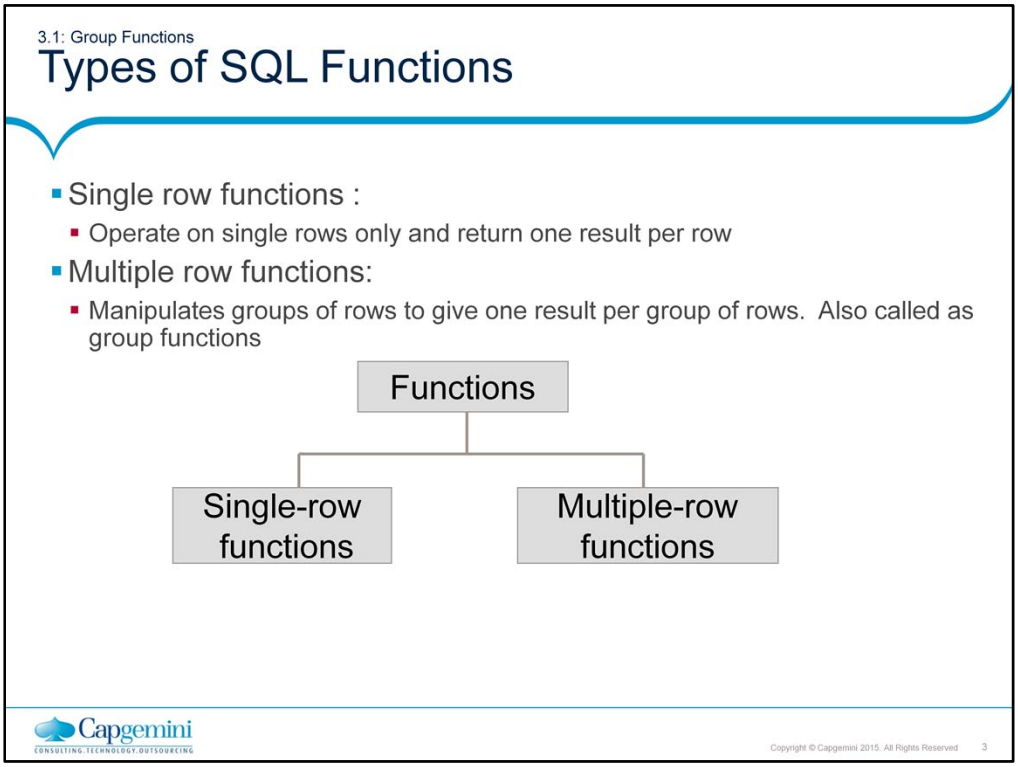
ORACLE SQL

Lesson 03: Group functions, Joins, Sub queries, set operators.

Lesson Objectives

- To understand the following topics:
  - Introduction to Functions
  - Aggregate (Group) functions:
    - GROUP BY clause
    - HAVING clause
  - Joins
    - Oracle Proprietary Joins
    - SQL: 1999 Compliant Joins
  - Types of joins
  - Sub-queries
  - Use of Set Operations



Functions:
Functions can be used to manipulate data values in a variety of ways.

Functions help in making the basic query block more powerful.

Functions may be used to perform calculations on data, alter data formats for display, convert data types, etc.

Functions are similar to operators in that they manipulate data items and return a result.

Functions differ from operators in the format in which they appear with their arguments. This format allows them to operate on zero, one, two, or more arguments:

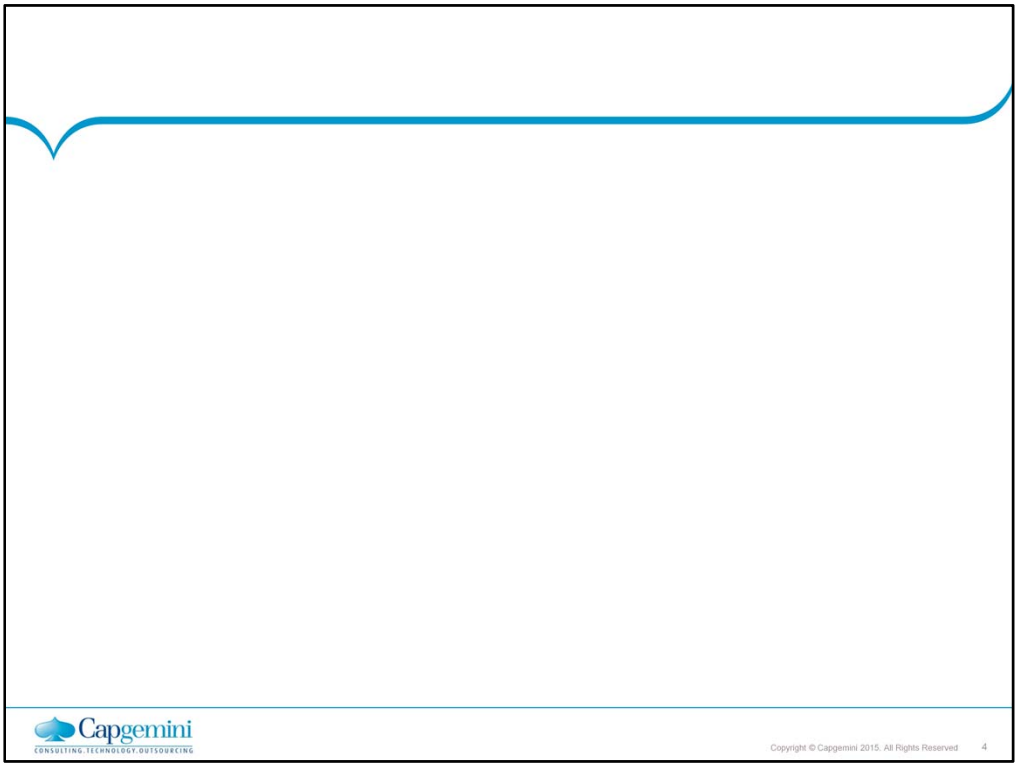
function(argument, argument, ...)

There are two types of functions: SQL functions and User-defined functions.

SQL functions: SQL functions are built into most DBMS and are available for use in various appropriate SQL statements. SQL functions are further categorized as:

Single-Row functions: Single-row functions return a single result row for every row of a queried Table or View.

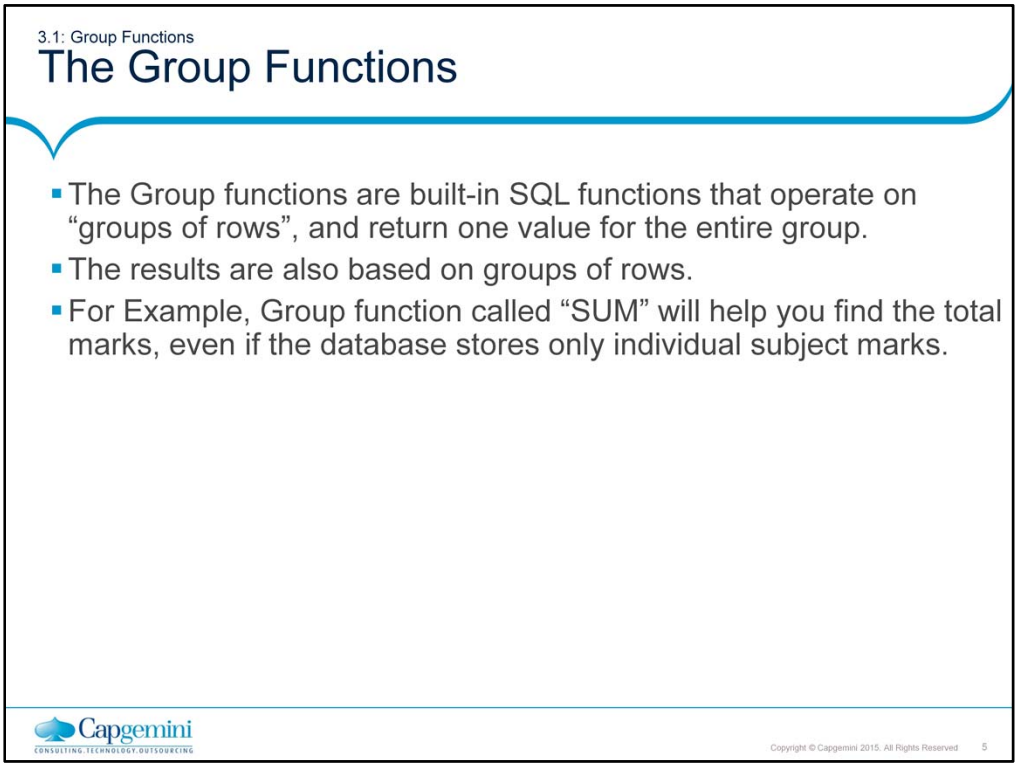
For example: ABS, ROUND etc.



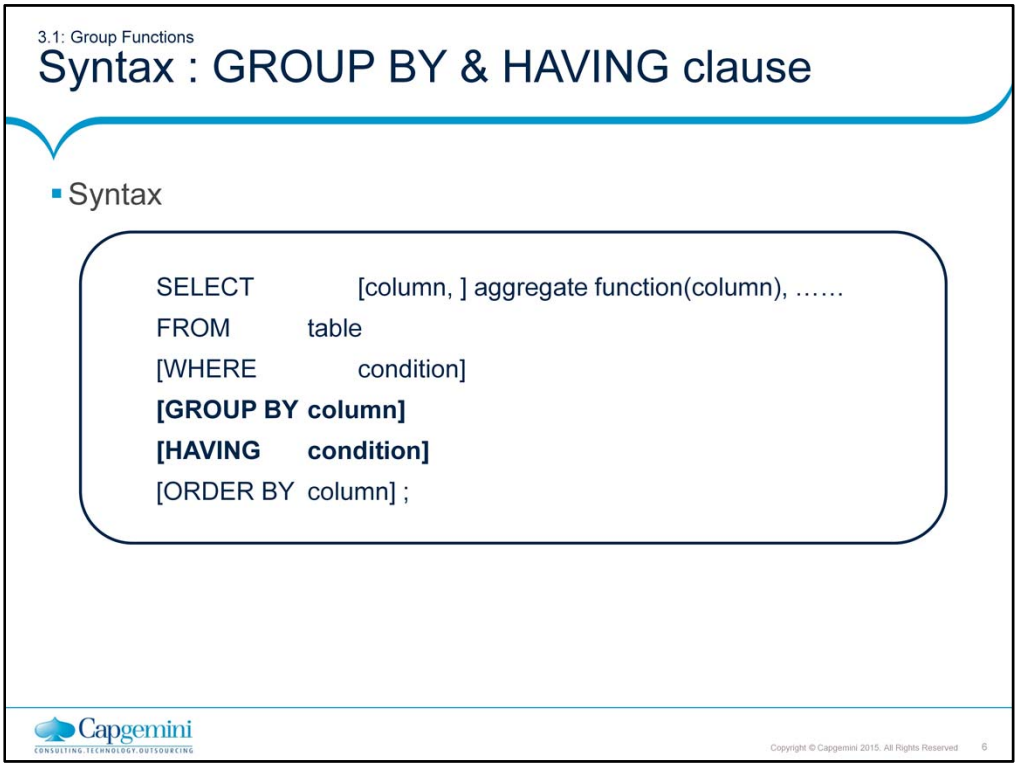
Aggregate functions: Aggregate functions return a single row based on groups of rows, rather than on single rows. For example: MAX, MIN, COUNT, etc.

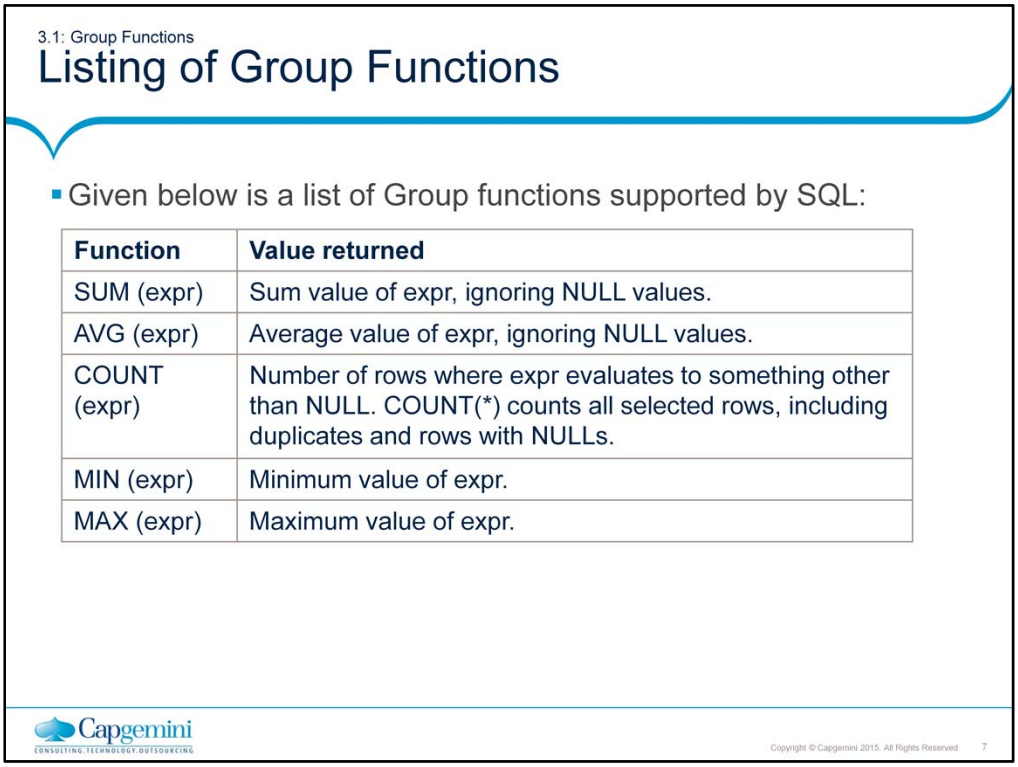
User-defined functions: You can write user-defined functions in PL/SQL or Java to provide functionality that is not available in SQL or SQL functions. User-defined functions can appear in a SQL statement wherever SQL functions can appear, that is, wherever an expression can occur.

For example: User-defined functions can be used in the following:
The select list of a SELECT statement
The condition of a WHERE clause
CONNECT BY, START WITH, ORDER BY, and GROUP BY clauses
The VALUES clause of an INSERT statement
The SET clause of an UPDATE statement



Aggregate (Group) Functions:
SQL provides a set of "built-in" functions for producing a single value for an entire group. These functions are called as "Set functions" or "Aggregate (Group) functions".
These functions can work on a "normal result table" or a "grouped result table".
If the result is not grouped, then the aggregate will be taken for the whole result table.


3.1: Group Functions
Syntax : GROUP BY & HAVING clause
Syntax
SELECT [column,] aggregate function(column), ……
FROM table
[WHERE condition]
[GROUP BY column]
[HAVING condition]
[ORDER BY column] ;
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
6



Aggregate (Group) Functions supported by SQL:
All the above functions operate on a number of rows (for example, an entire table), and are therefore known as “Group (or Aggregate) functions”.
A Group function can be used on a subset of the rows in a table by using the WHERE clause.
The Aggregate functions ignore NULL values in the column.

To include NULL values, NVL function can be used with Aggregate functions.

Note :
Count(*): Returns the number of rows in the table, including duplicates and those with NULLs.
Count(<Expression>): Returns the number of rows where expression is NOT NULL.

Aggregate (Group) Functions supported by SQL (contd.):

SUM(COL_NAME | EXPRESSION)

- SUM returns the total of values present in a particular “column” or a “number of columns” that are linked together in the expression. All the columns, which form the argument to SUM, must be numeric only.
- To find the sum of one subject for all students following query is used:

```
SELECT SUM(subject1)
        FROM student_marks;
```

- To find the yearly compensation of staff from department 20, the following query is used:

```
SELECT SUM(12*staff_sal)
        FROM staff_master
        WHERE dept_code = 20;
```

AVG(COL_NAME | EXPRESSION)

- AVG is similar to SUM. AVG returns the average of a NUMBER of values. The restrictions, which apply on SUM also, apply on AVG.
- To find the average salary the staff, the following query is used:

```
SELECT AVG(sal)
        FROM staff_master;
```

- To find the average yearly compensation of staff from department 20, the following query is used:

```
SELECT AVG(12*staff_sal)
        FROM staff_master
        WHERE dept_code = 20;
```

Aggregate (Group) Functions supported by SQL (contd.):

COUNT(*)

- COUNT returns the number of rows.
- To find the total number of staff members, the following query is used:

```
SELECT COUNT(*)
    FROM staff_master;
```

  - It is possible to restrict the rows for the operation of COUNT.
- To find the total number of staff members in department 10, the following query is used:

```
SELECT COUNT(*)
    FROM staff_master
    WHERE dept_code = 10 ;
```

- To find the total number of staff members hired after '01-JAN-02', the following query is used:

```
SELECT COUNT(*)
    FROM staff_master
    WHERE hiredate >'01-JAN-02' ;
```

  - When an aggregate function is used in a SELECT statement, column names cannot be used in SELECT unless GROUP BY clause is used.

MIN(COL_NAME | EXPRESSION)

- MIN returns the lowest of the values from the column. MIN accepts columns, which are NON-NUMERIC too.
- To find the minimum salary paid to a staff member, the following query is used:

```
SELECT MIN(staff_sal)
    FROM staff_master;
```

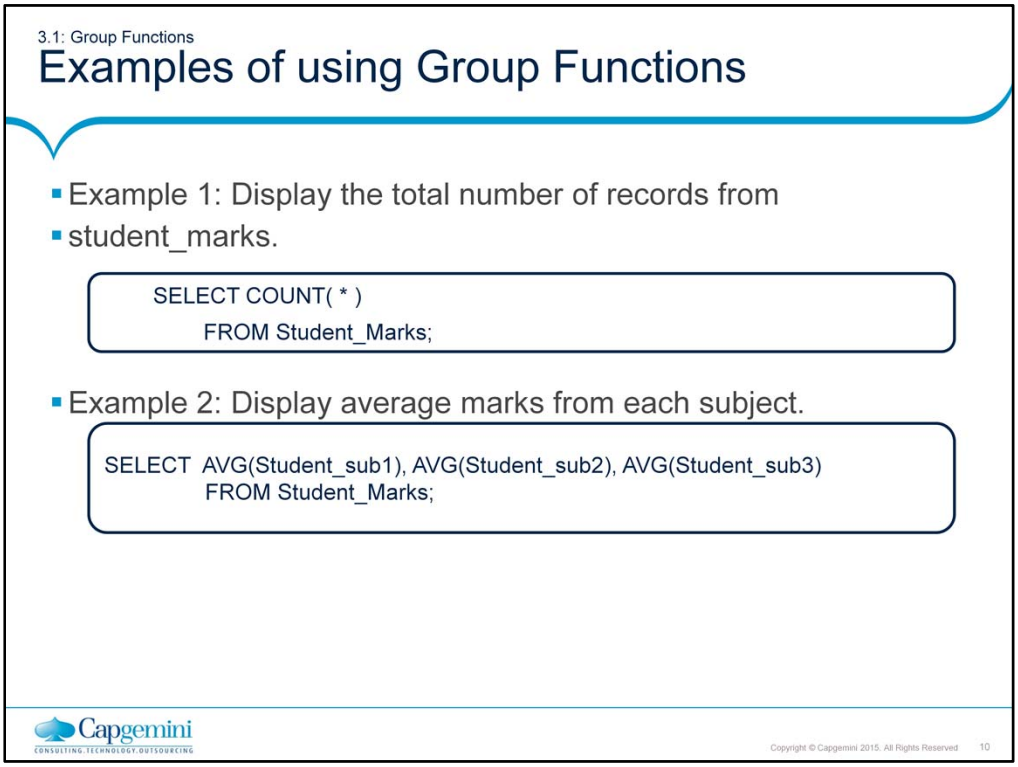
- To list the staff member who alphabetically heads the list, the following query is used:

```
SELECT MIN(staff_name)
    FROM staff_master;
```

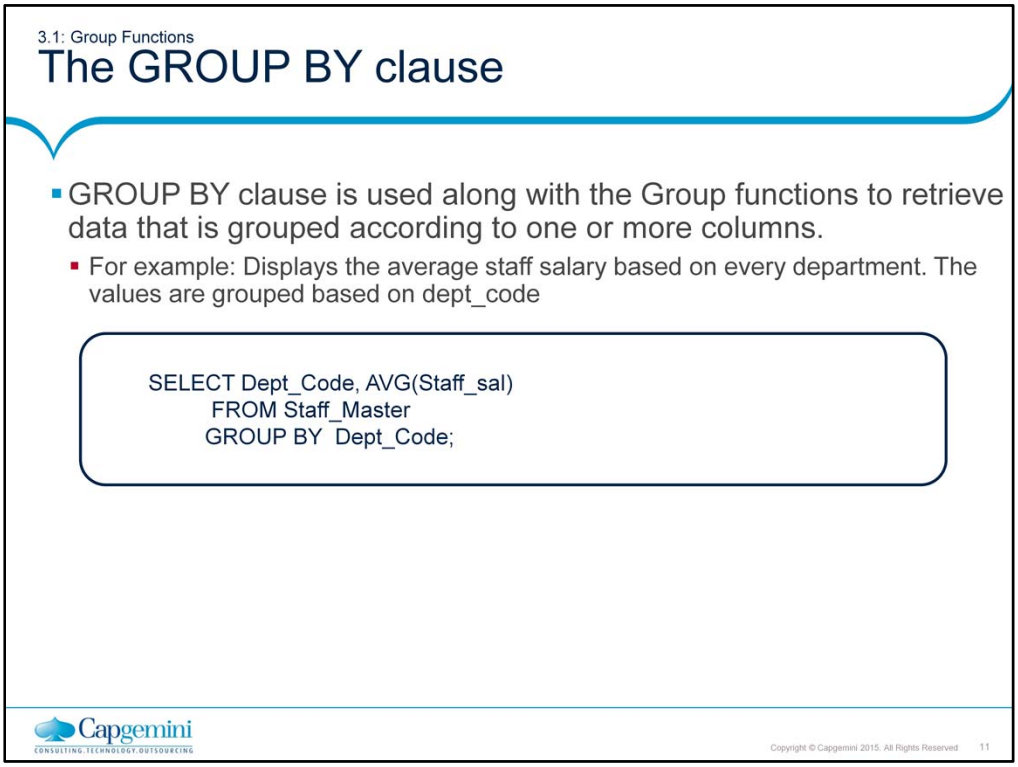
MAX(COL_NAME | EXPRESSION)

- MAX is the reverse of MIN. MAX returns the maximum value from among the list of values.
- To find the maximum marks for a student in subject2, the following query is used:

```
SELECT MAX(subject2)
    FROM student_marks ;
```



Note:
The first query returns the value, which counts the number of rows fetched from the student_marks table. All the rows in thetable are treated as one group.Group Functions operate on sets of rows to give one result per group. Can be used on the whole table or certain set of rows

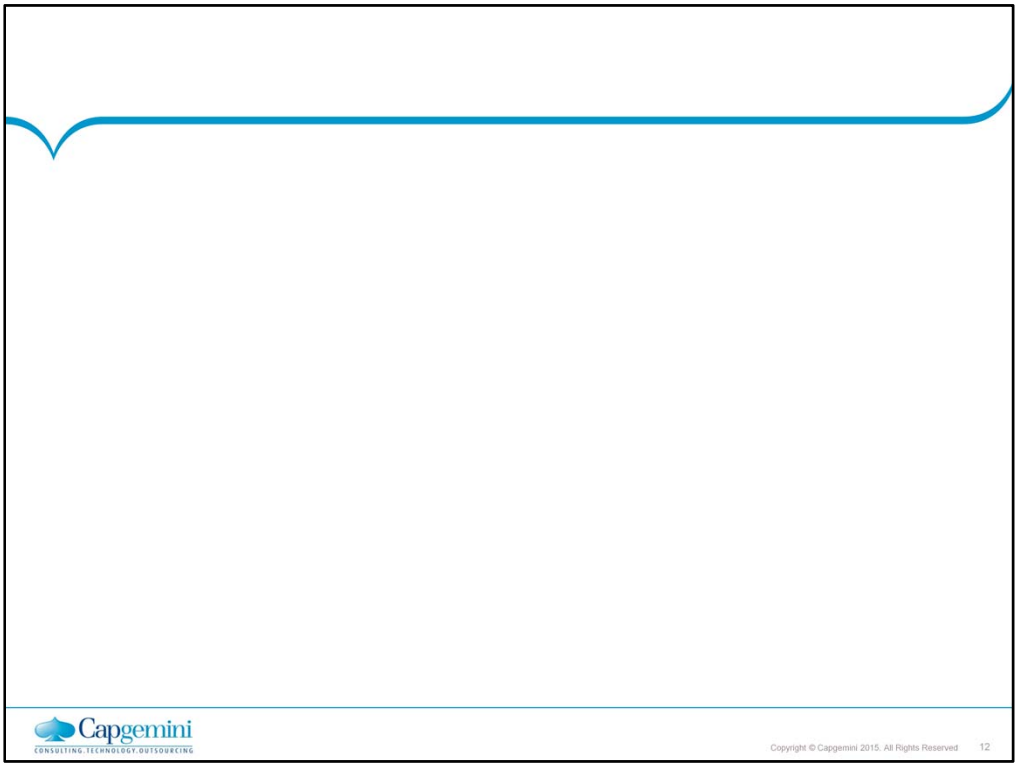


Usage of GROUP BY and HAVING clauses:
All the SELECT statements we have used until now have acted on data as if the data is in a “single group”. But the rows of data in some of the tables can be thought of as being part of “different groups”.

For example: The staff_master table contains staff information allocated to various departments identified by dept_code. If we wish to find the minimum salary of each group of employees in respective department, then none of the clauses that we have seen until now are of any use.

The GROUP BY clause is used to group the result table derived from earlier FROM and WHERE clauses. Further, a HAVING clause is used to apply search condition on these groups.

When a GROUP BY clause is used, each row of the resulting table will represent a group having same values in the column(s) used for grouping. Subsequently, the HAVING clause acts on the resulting grouped table to remove the row that does not satisfy the criteria in the HAVING search condition



The GROUP BY Clause:

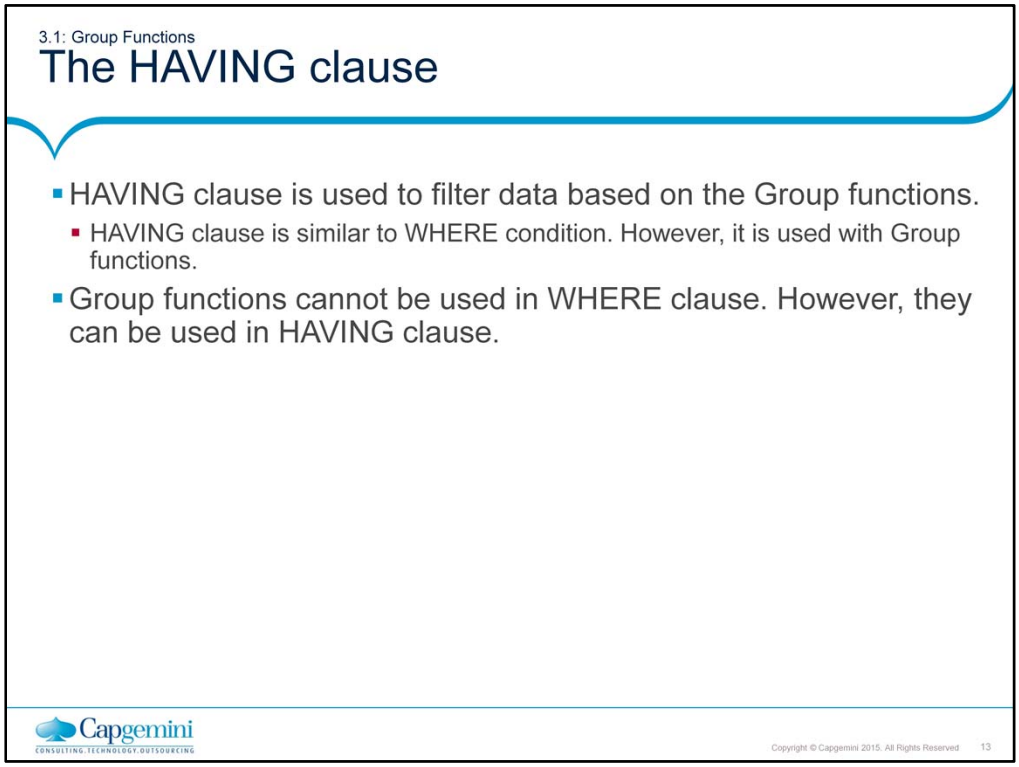
If you have to fetch columns other than group functions or if you want your results to be grouped on certain criteria use “GROUP BY” clause

- The GROUP BY clause is of the form:

```
GROUP BY <column  list>
```

- The columns specified must be selected in the query. If a table is grouped, the SELECT list should have columns and expressions, which are single-valued for a group. These can be:
  - columns on which grouping is done
  - constants
  - aggregate functions on the other columns on which no grouping is done

1. The GROUP BY clause should contain all the columns in the SELECT list, except those used along with the Group functions. Only the column names which have been used in GROUP BY clause and aggregate columns can be used in SELECT clause
2. When an Aggregate (Group) function is used in a SELECT statement, the column names cannot be used in SELECT, unless GROUP BY clause is used.
3. Grouping can be done on multiple columns, as well.



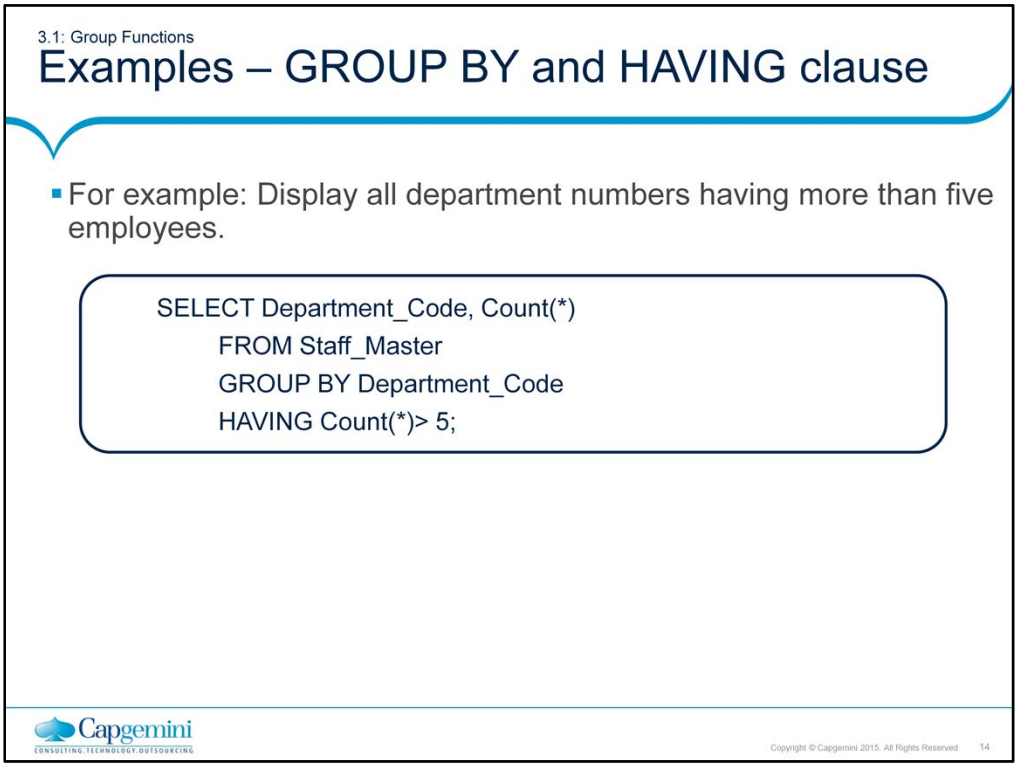
The HAVING Clause:
A HAVING clause is of the form:

```
HAVING <search condition>
```

The HAVING search condition applies to “each group”. It can be:
- formed using various predicates like between, in, like, null, comparison, etc
- combined with Boolean operators like AND, OR, NOT

Since the search condition is for a “grouped table”. The predicates should be:
- on a column by which grouping is done.
- on a set function (Aggregate function) on other columns.

The aggregate functions can be used in HAVING clause. However, they cannot be used in the WHERE clause.
When WHERE, GROUP BY, and HAVING clauses are used together in a SELECT statement:
- The WHERE clause is processed first in order.
- Subsequently, the rows that are returned after the WHERE clause is executed are grouped based on the GROUP BY clause.
- Finally, any conditions, on the Group functions in the HAVING clause, are applied to the grouped rows before the final output is displayed.

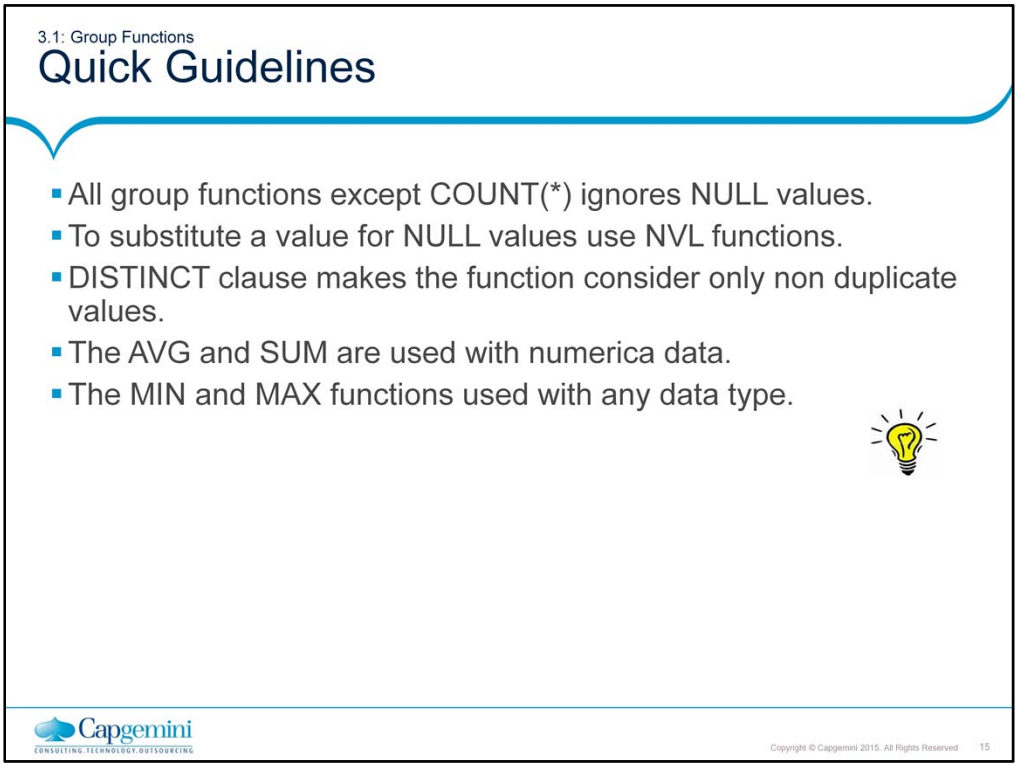


The HAVING clause (contd.):
To find out Average, Maximum, Minimum salary of departments, where average salary is greater than 2000.

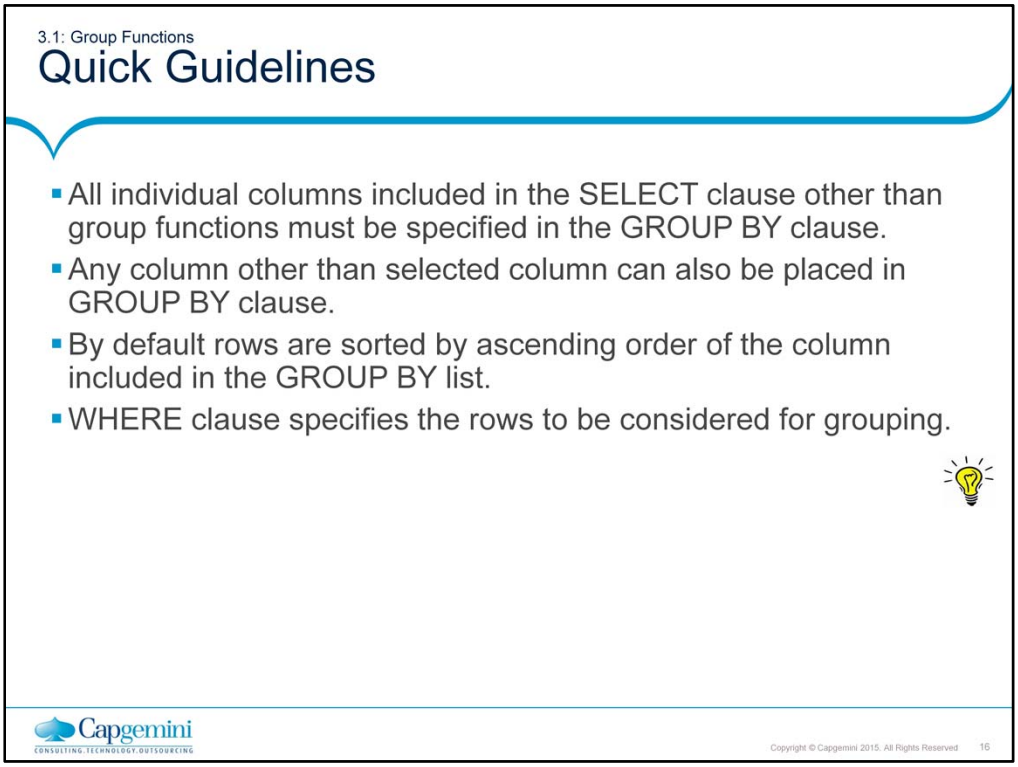
```
SELECT dept_code,AVG(staff_sal),MIN(staff_sal),
MAX(staff_sal)
FROM staff_master
GROUP BY dept_code HAVING AVG(staff_sal) > 2000;
```

To find out average salary of all staff members who belong to department 10 or 20 and their average salary is greater than 10000

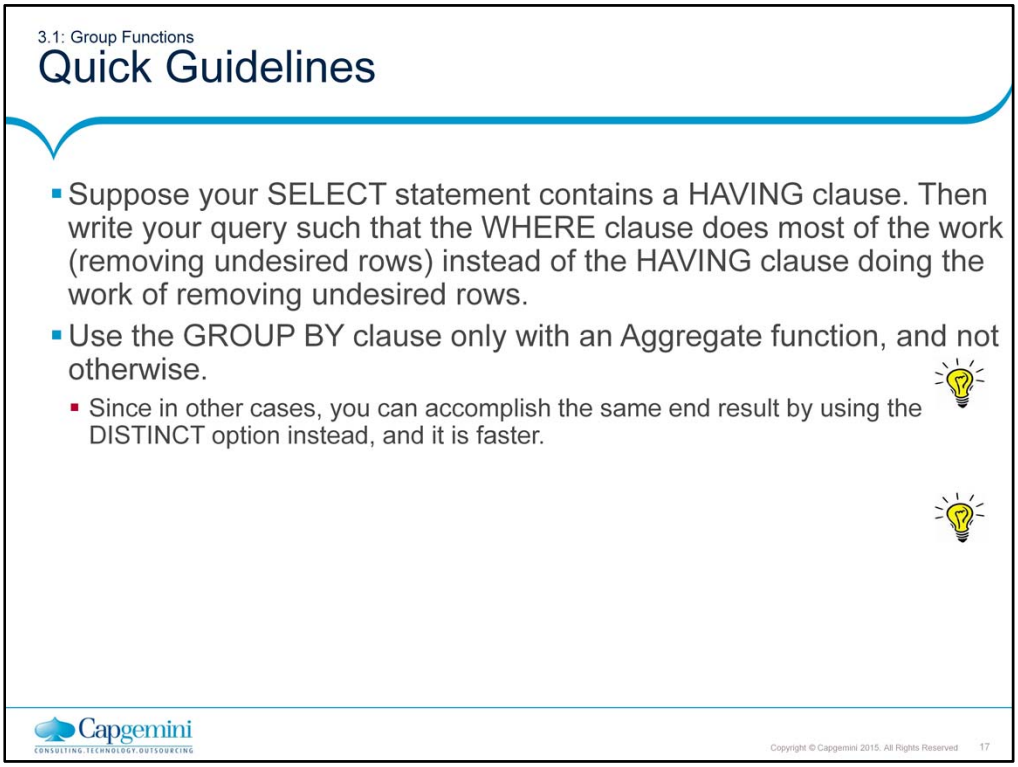
```
SELECT design_code,dept_code,avg(staff_sal)
FROM staff_master where dept_code in(10,20)
GROUP BY  design_code,dept_code
HAVING avg(staff_sal) >10000;
```



NVL function is covered in the next lesson



NVL function is covered in the next lesson



Tips and Tricks:
By appropriately using the WHERE clause, you can eliminate unnecessary rows before they reach the GROUP BY and HAVING clause. Thus saving some unnecessary work, and boosting performance.

For example: In a SELECT statement with WHERE, GROUP BY, and HAVING clauses, the query executes in the following sequence.

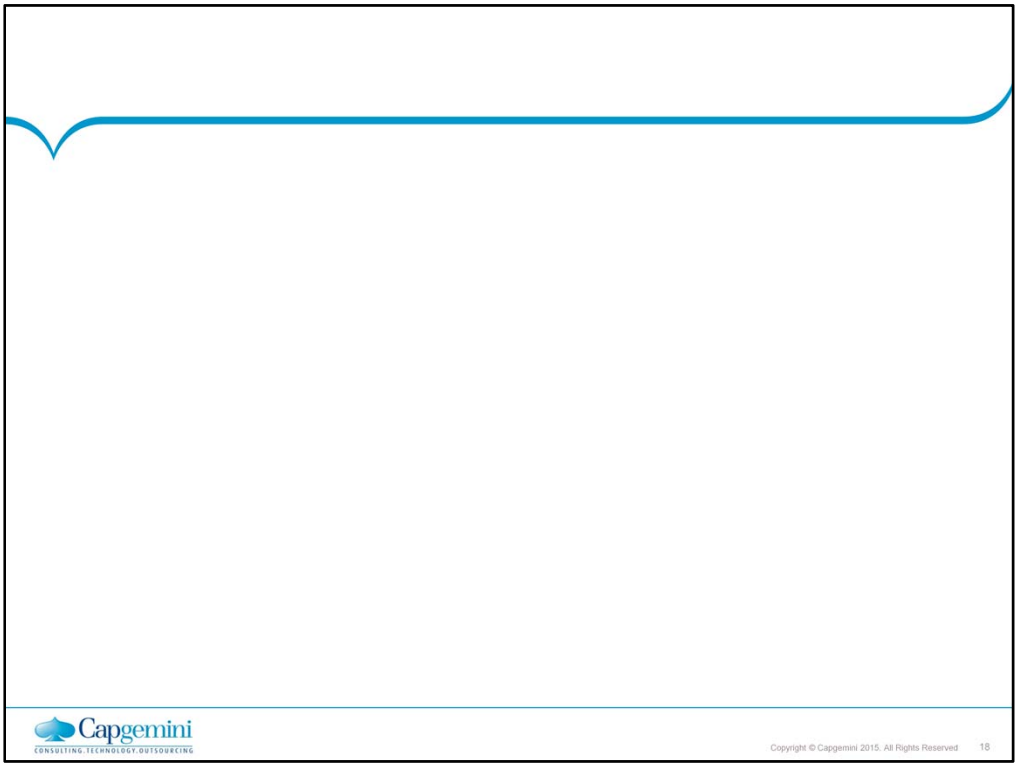
First, the WHERE clause is used to select the appropriate rows that need to be grouped.

Next, the GROUP BY clause divides the rows into sets of grouped rows, and then aggregates their values.

And last, the HAVING clause then eliminates undesired aggregated groups.

If the WHERE clause is used to eliminate as many of the undesired rows as possible, then the GROUP BY and the HAVING clauses will have to do less work. Thus boosting the overall performance of the query.

contd.



Tips and Tricks (contd.):

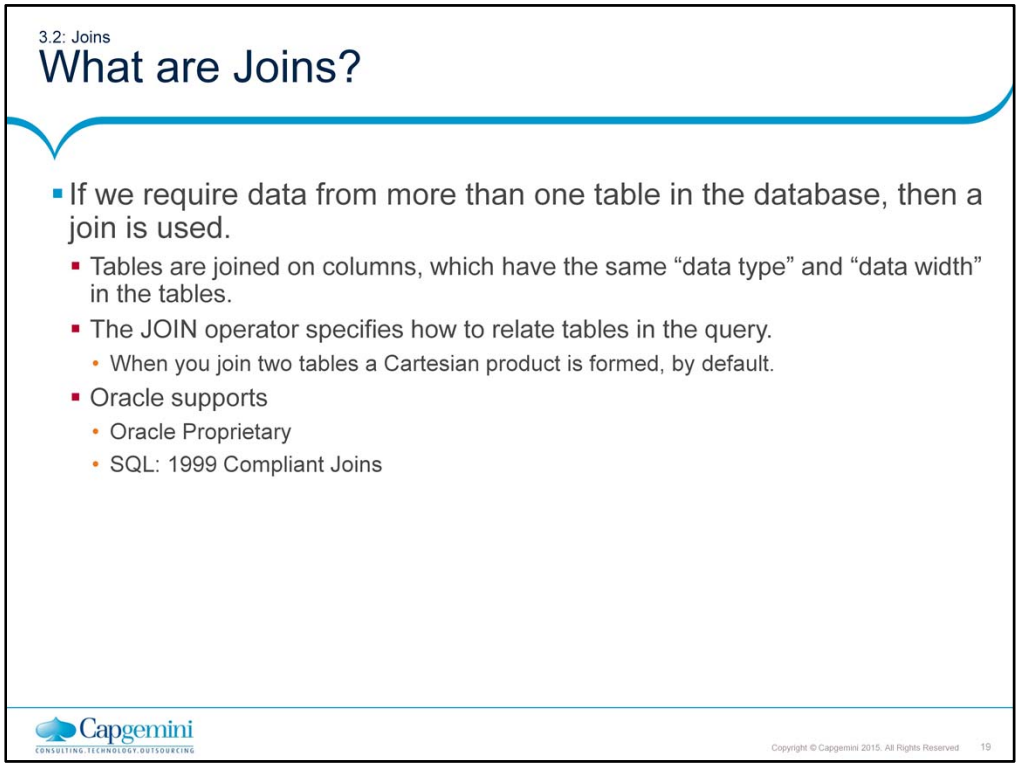
- The GROUP BY clause can be used with or without an Aggregate function. However, if you want optimum performance, do not use the GROUP BY clause without an Aggregate function. This is because you can accomplish the same end result by using the DISTINCT option instead, and it is faster.
  For example: You can write your query two different ways:

```
SELECT OrderID
FROM [Order Details]
WHERE UnitPrice > 10
GROUP BY OrderID
```

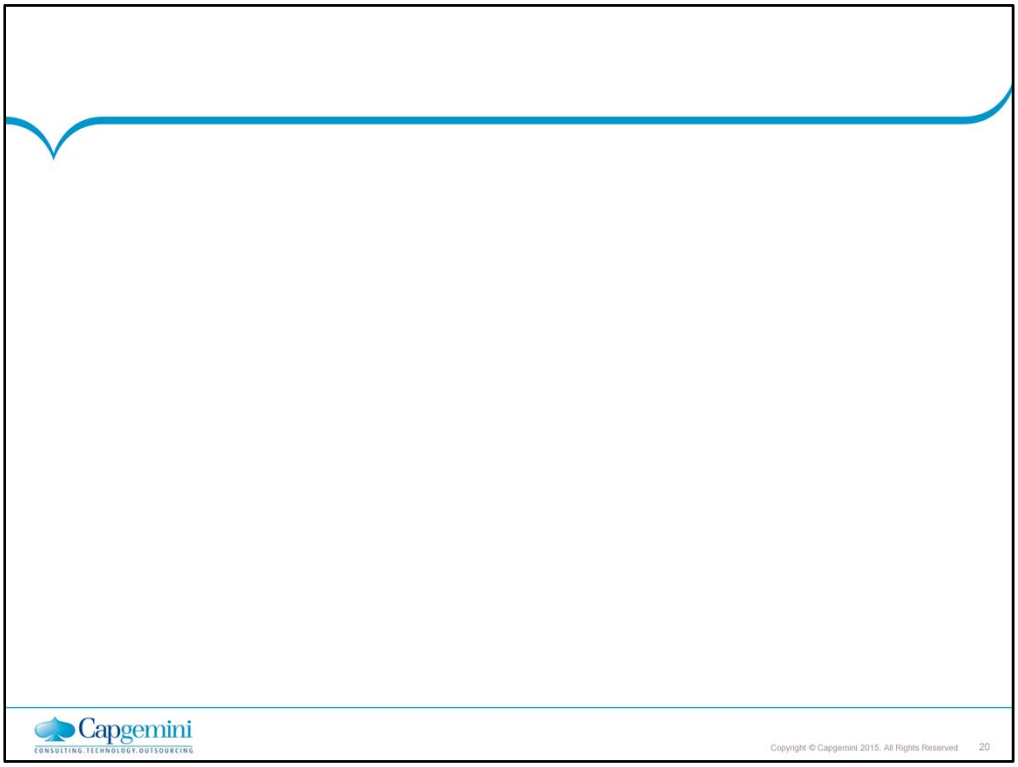
or

```
SELECT DISTINCT OrderID
FROM [Order Details]
WHERE UnitPrice > 10
```

Both of the above queries produce the same results, but the second one will use less resources and perform faster.



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:

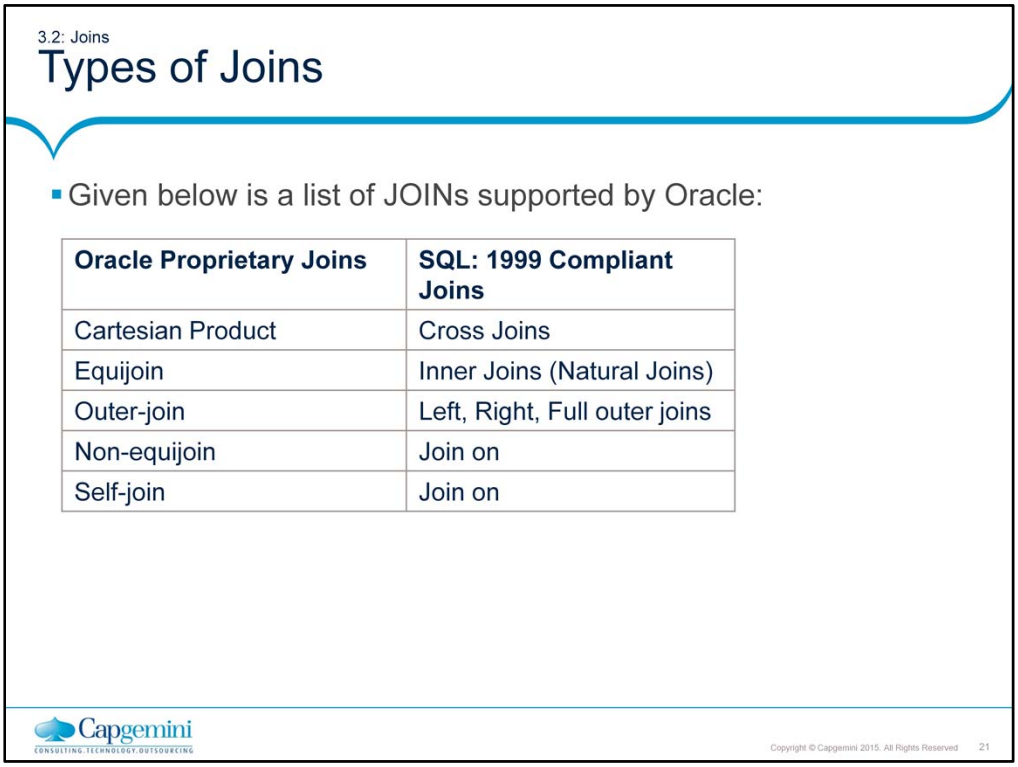
COL1	COL2	COL1	COL2
A	1	X	100
B	2	X	100
C	3	X	100
A	1	Y	200
B	2	Y	200
C	3	Y	200

6 rows selected

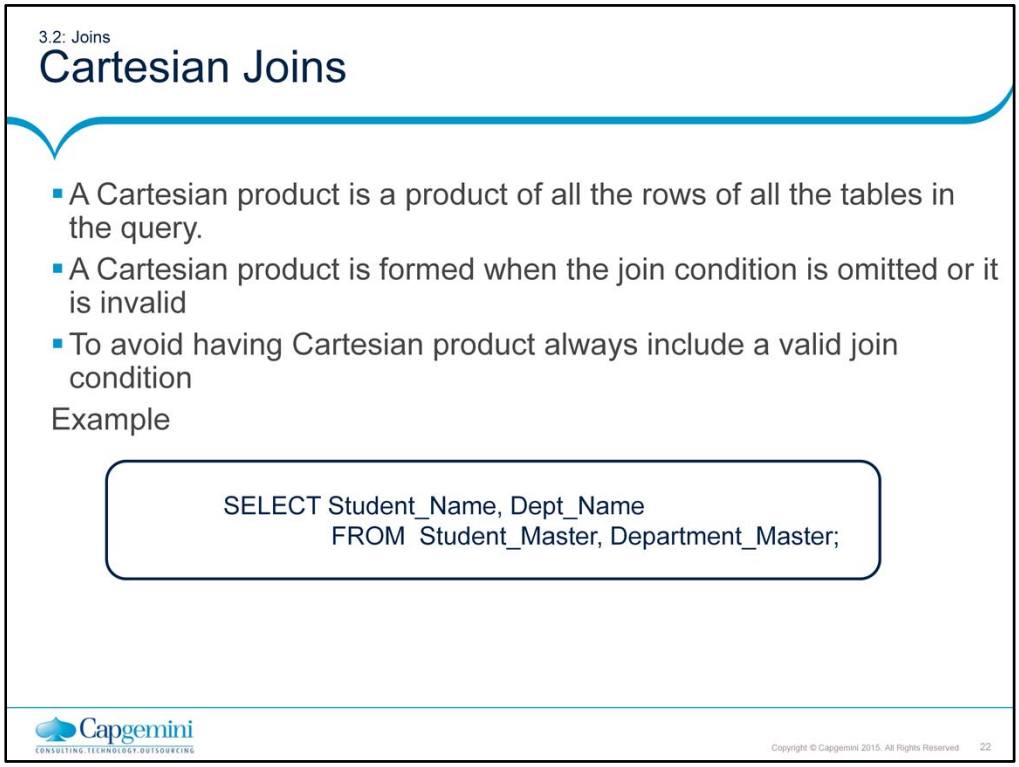
The statement SELECT * FROM F1 First_T, F1 Second_T; results in:

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

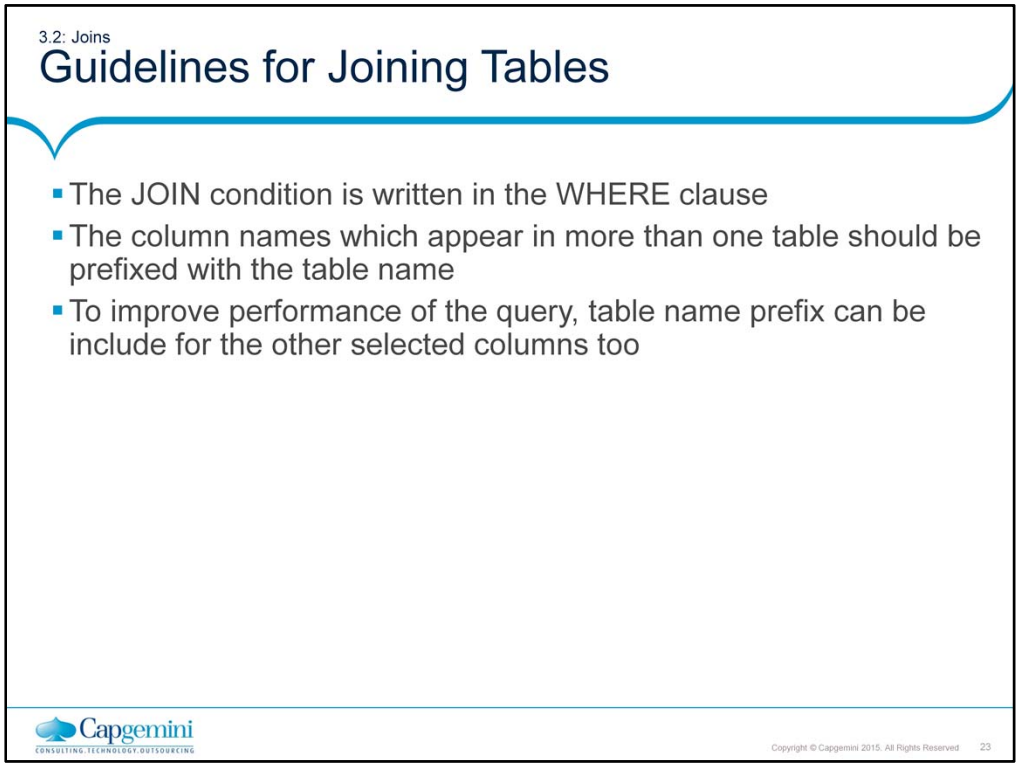
9 rows selected



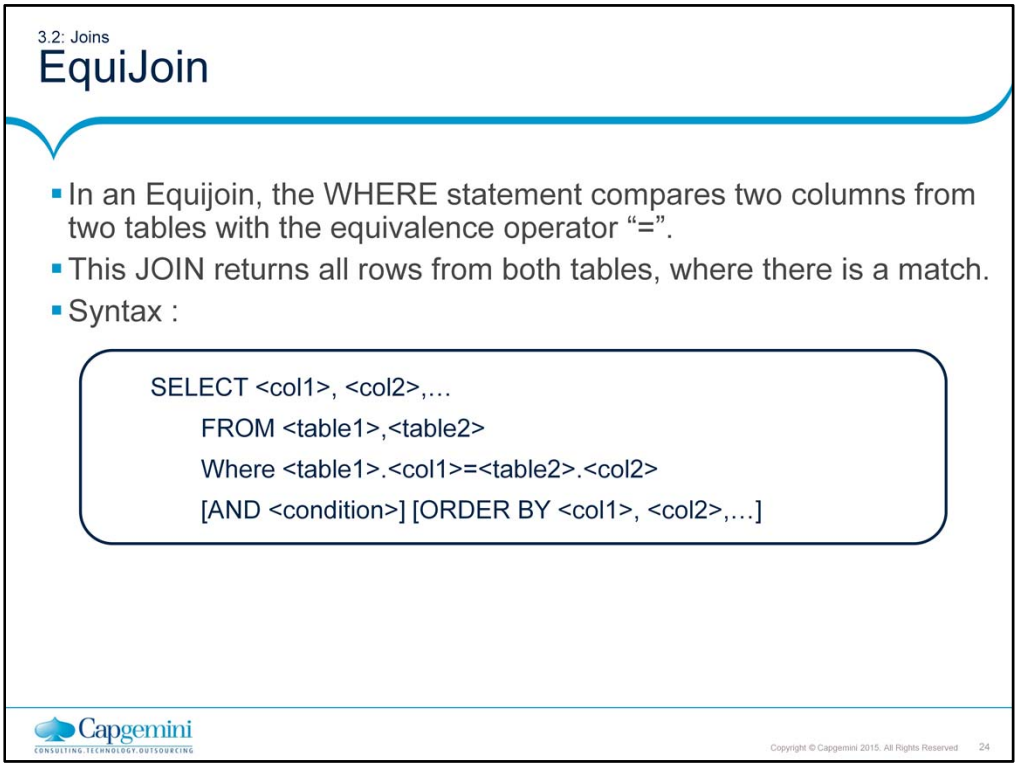
Note:
Oracle9i onwards offers JOIN syntax that is SQL: 1999 compliant.
Prior to the 9i release, the JOIN syntax was different from the ANSI standards.
The new SQL: 1999 compliant JOIN syntax does not offer any performance benefits over the Oracle proprietary JOIN syntax that existed in prior releases.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

3.2: 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;
```

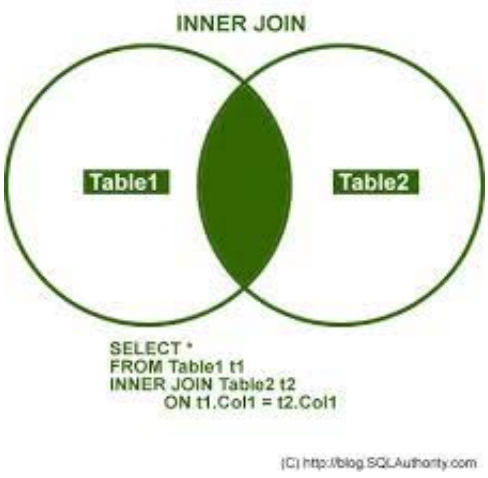
- Example 2: To display student and staff name along with the department name to which they belong

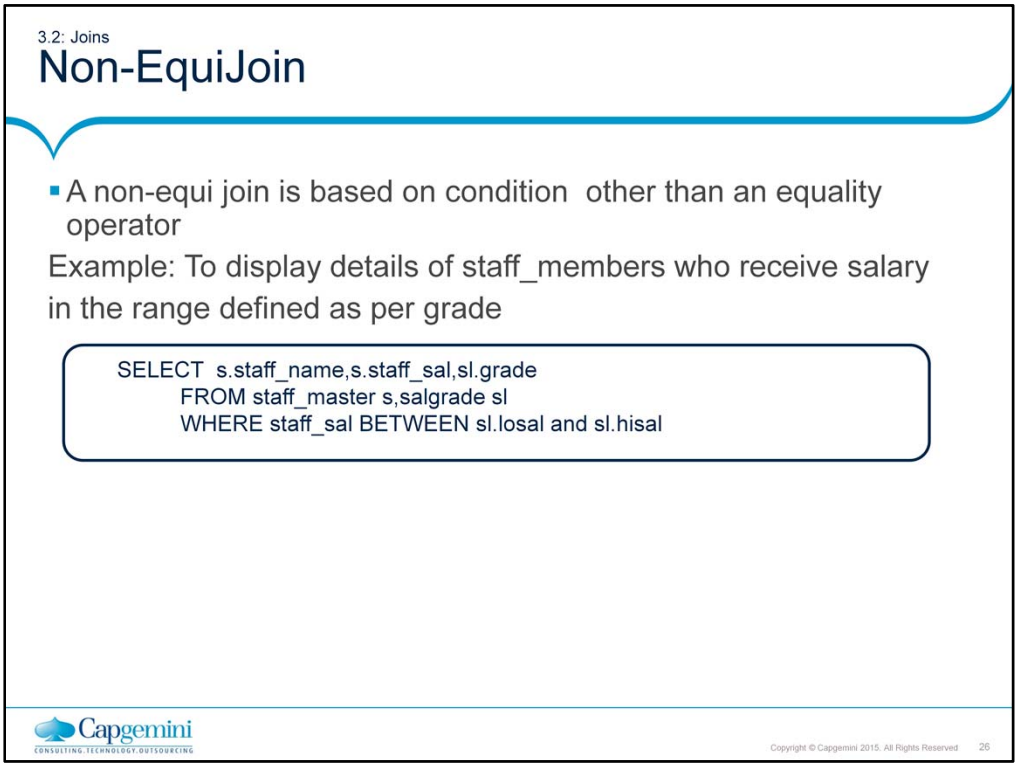
```
SELECT student_name,staff_name, dept_name
        FROM student_master, department_master,staff_master
        WHERE student_master.dept_code=department_master.dept_code
         and staff_master.dept_code=department_master.dept_code;
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 25

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

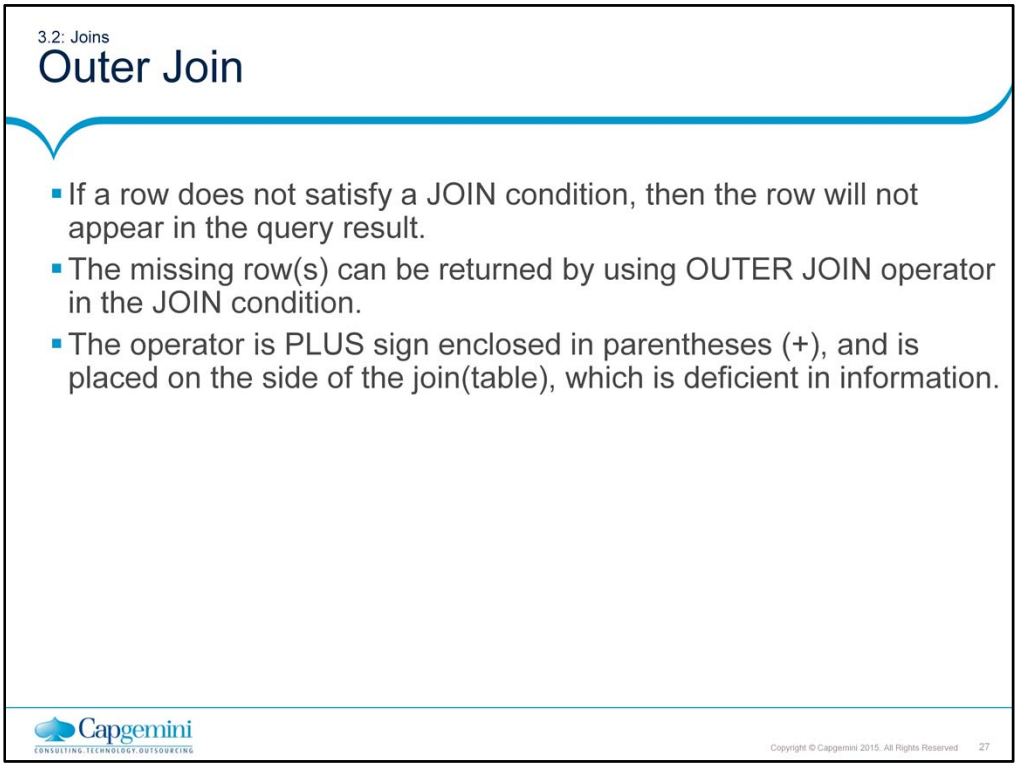




Non-Equijoin
A Non-equijoin is a JOIN condition containing something other than an equality operator.
The example on the slide shows a non-equijoin operation
Assume that we have a Salgrade table which is used to determine the range of salary for all staff member. The structure of the table is as follows:

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.

3.2: 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

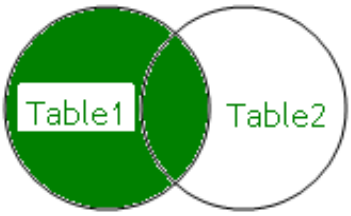
```
WHERE table1 <OUTER JOIN INDICATOR> = table 2
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

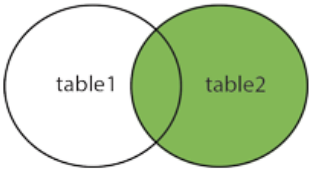
 28

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 outer join diagram :



RIGHT OUTER JOIN



3.2: Joins

Outer Join - Example

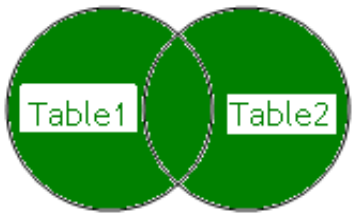
- To display Department details which have staff members and also display department details who do not have any staff members

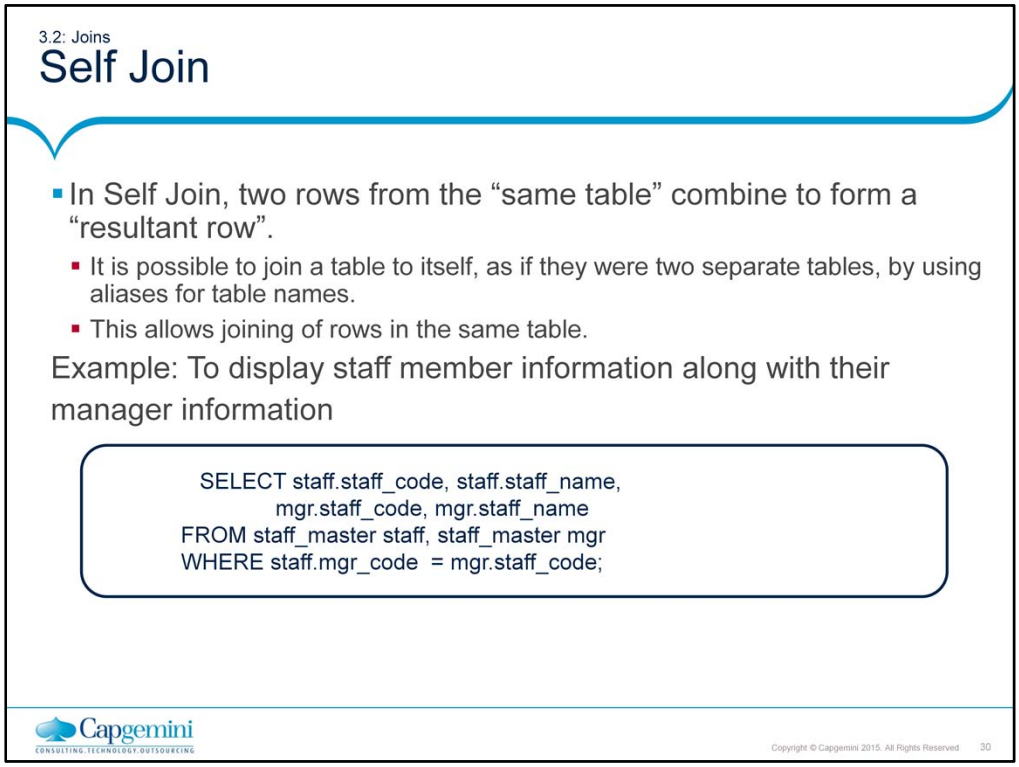
```
SELECT staff.staff_code,staff.Dept_Code,dept.Dept_name
  FROM Staff_master staff, Department_Master dept
  WHERE staff.Dept_Code(+) = dept.Dept_Code
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 29

Full outer join diagram



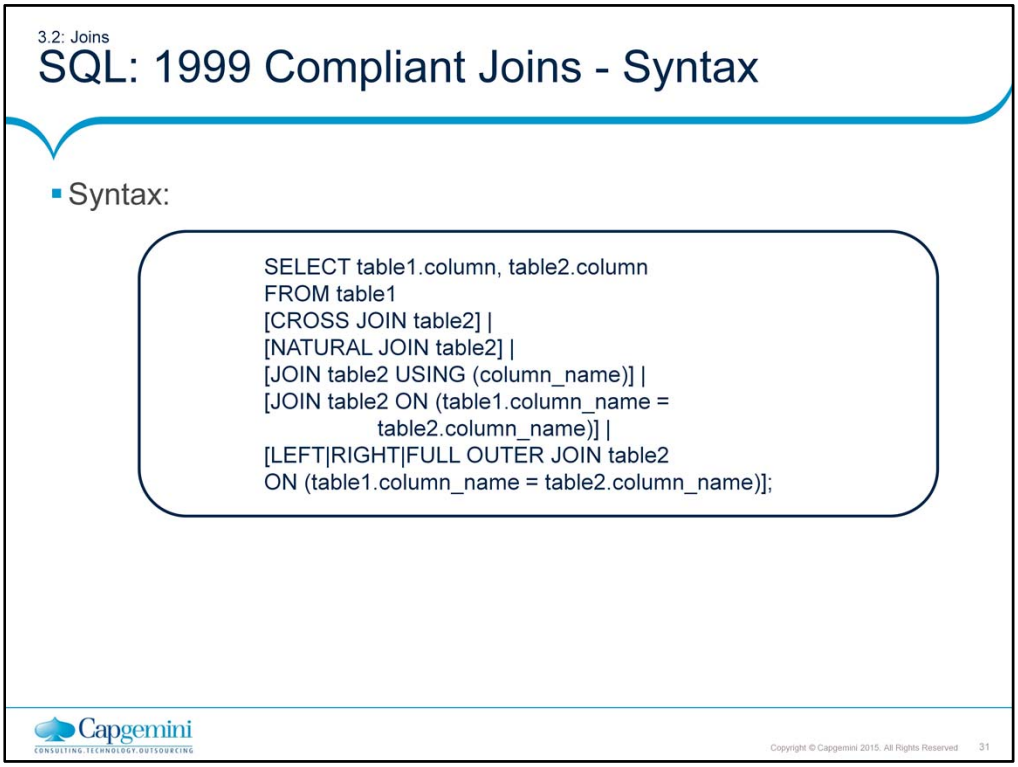


Self Join:
Sometimes is required to join the table to itself. To join a table to itself, “two copies” of the same table have to be opened in the memory.
Since the table names are the same, the second table will overwrite the first table. In effect, this will result in only one table being in memory.

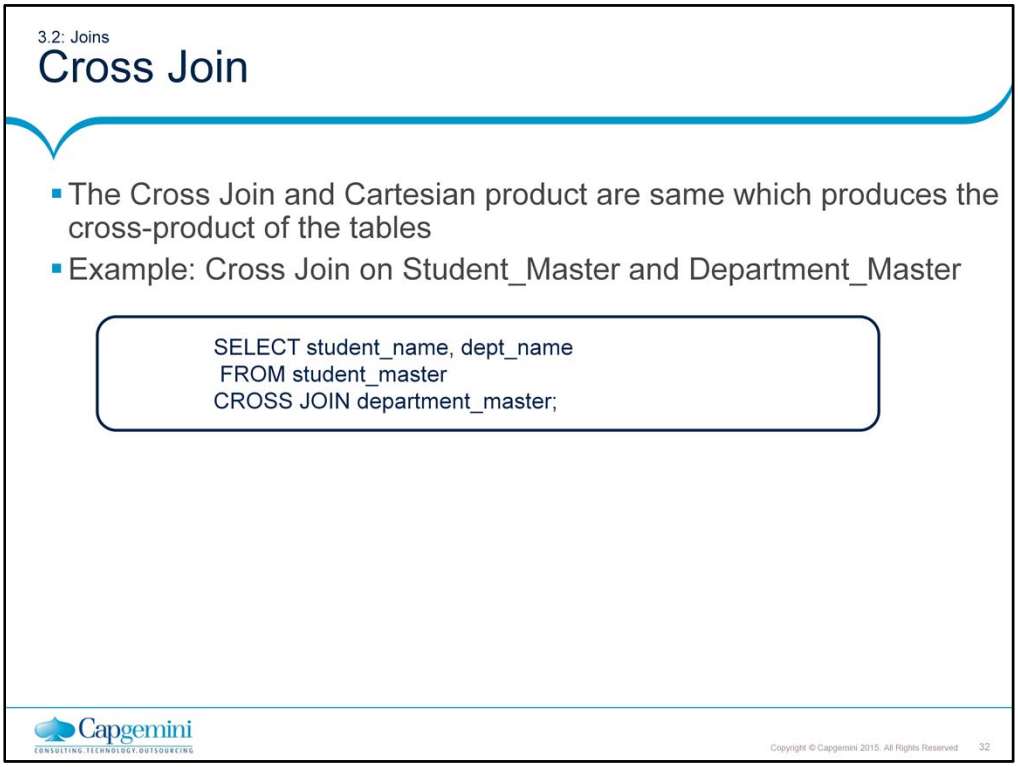
This is because a table name is translated into a specific memory location. Hence in the FROM clause, the table name needs to be mentioned twice with an “alias”

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.



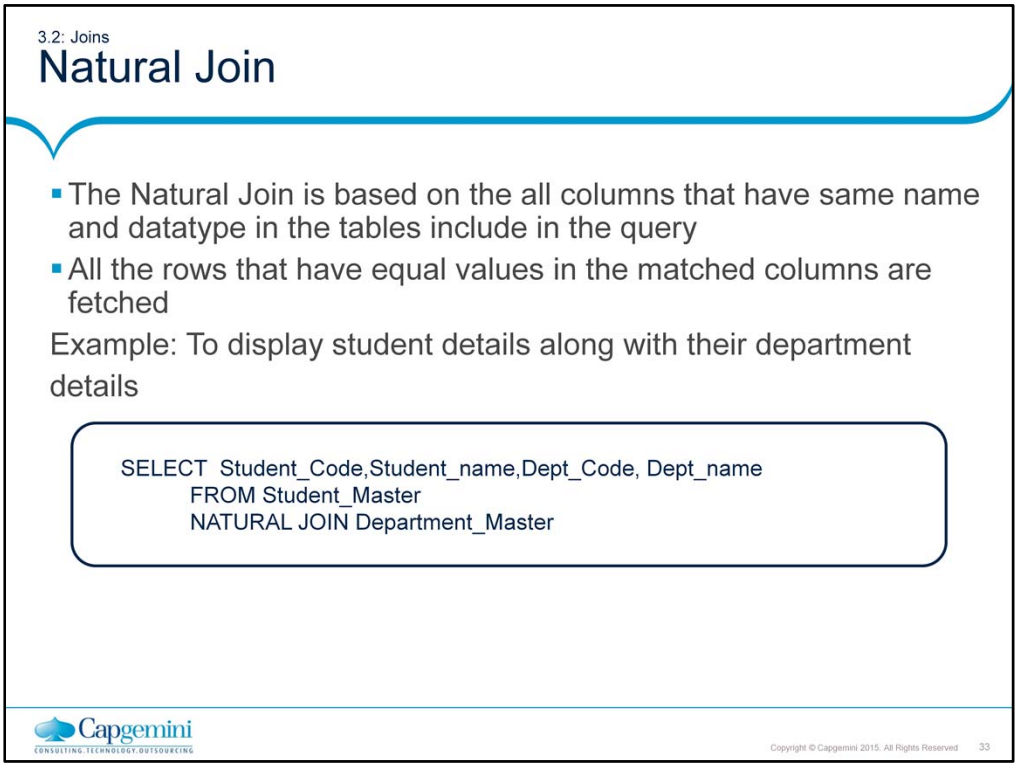
SQL:1999 Compliant Joins
The SQL Compliant joins can obtain the similar results that we have discussed in the previous slides. They differ only in syntax.
The SQL compliant joins are supported from Oracle 9i version onwards



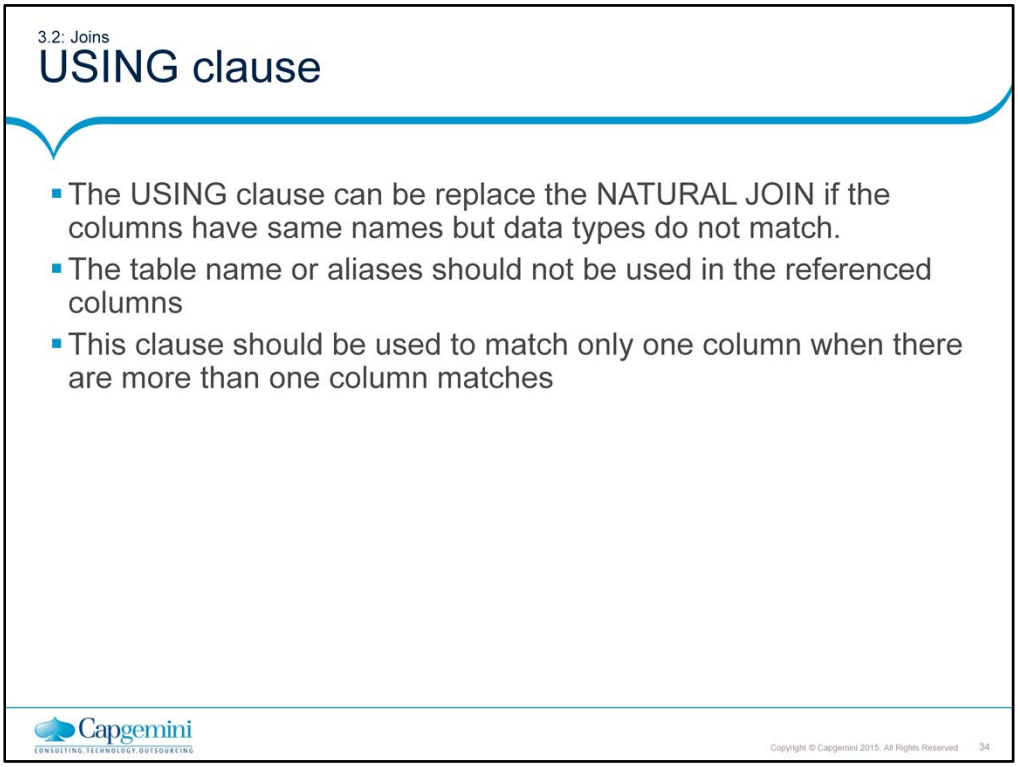
Cross Join

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

3.2: Joins

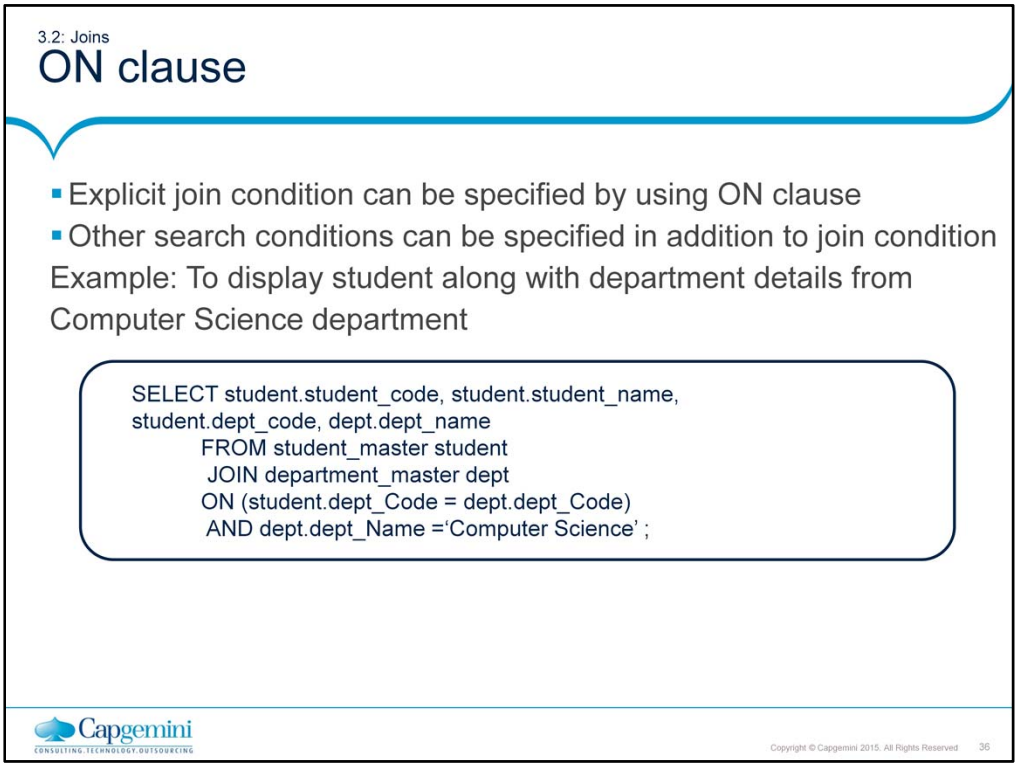
USING clause - Example

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

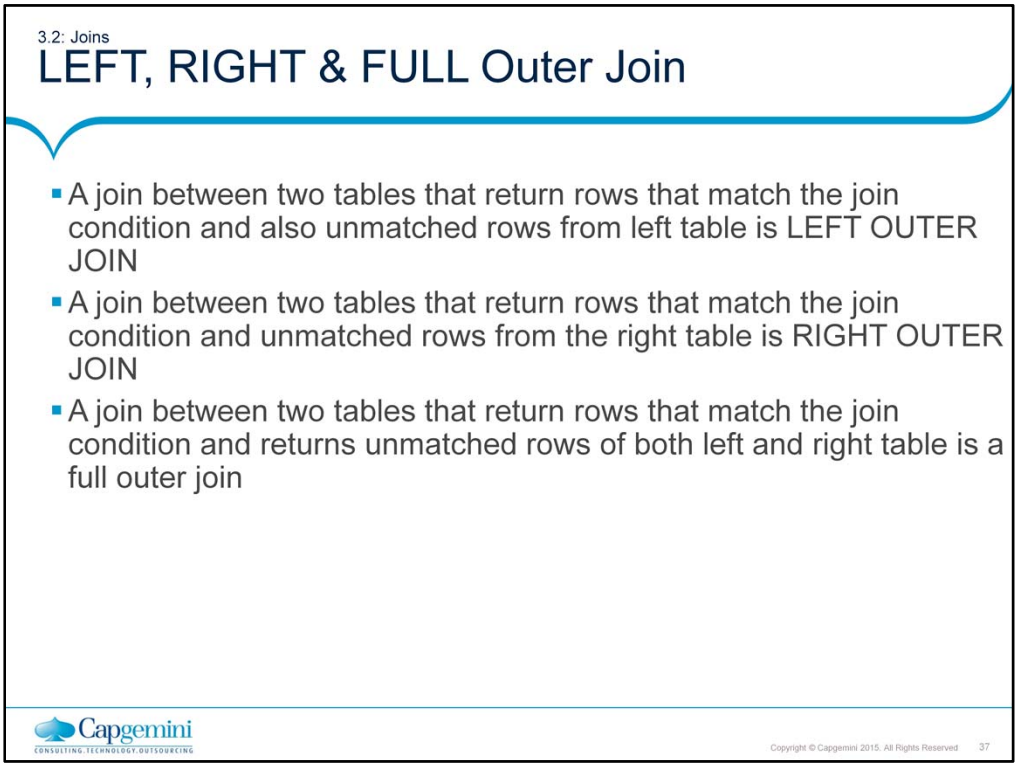
 35



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

3.2: Joins

LEFT, RIGHT & FULL Outer Join - Example

- Example 1: Display student & department details and also those 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

 38

3.2: Joins

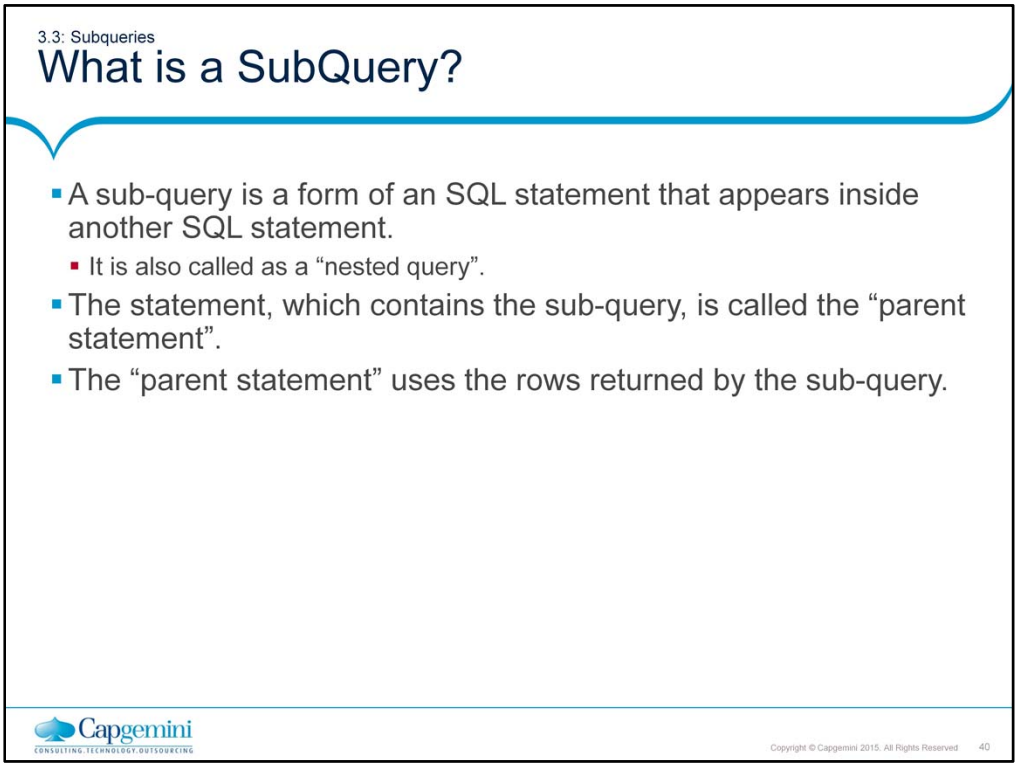
LEFT, RIGHT & FULL Outer Join - Example

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

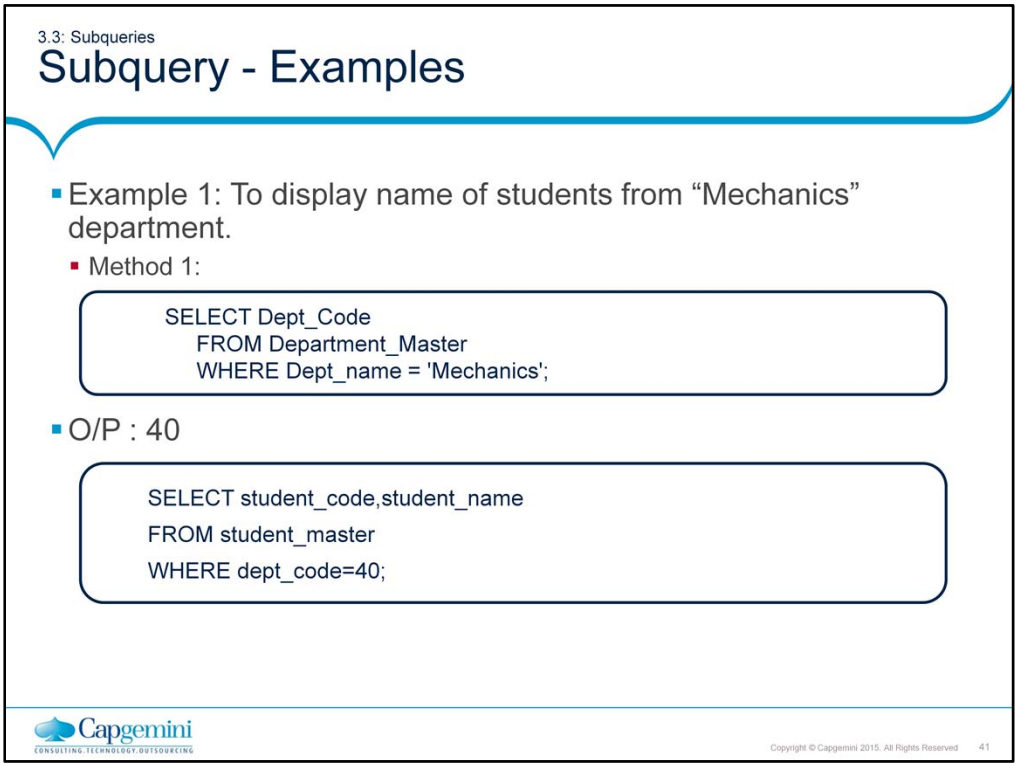
```
SELECT s.student_code,s.dept_code,d.dept_name
        FROM student_master s
        FULL OUTER JOIN department_master d
        ON (s.dept_code = d.dept_code );
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

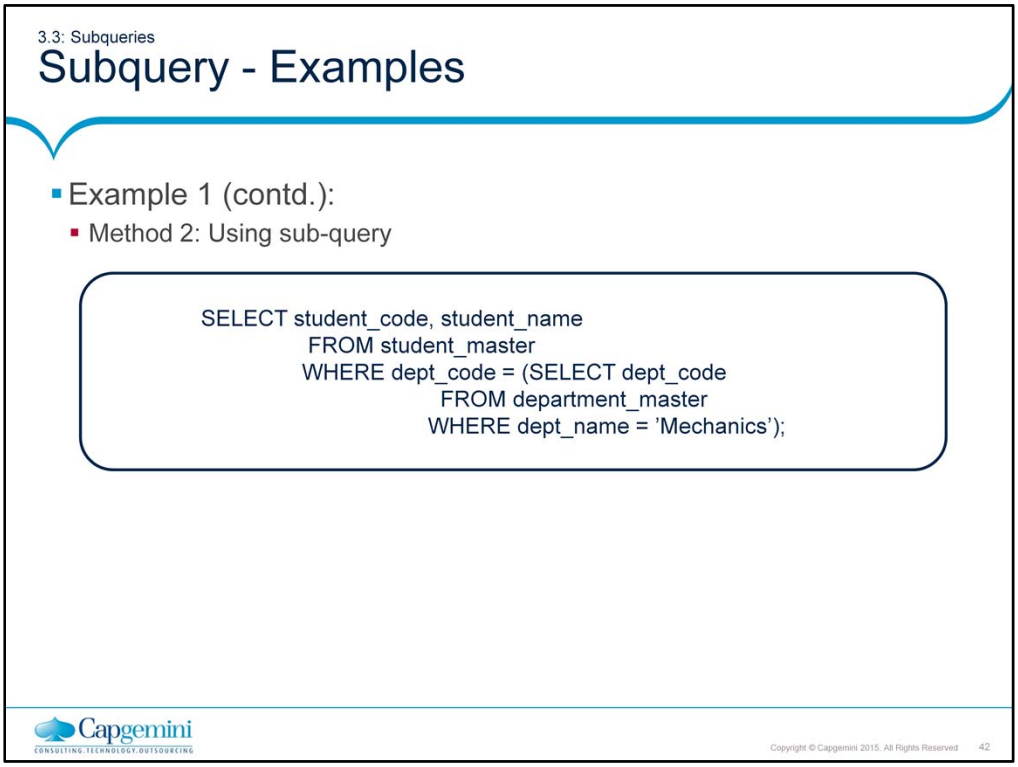
 39



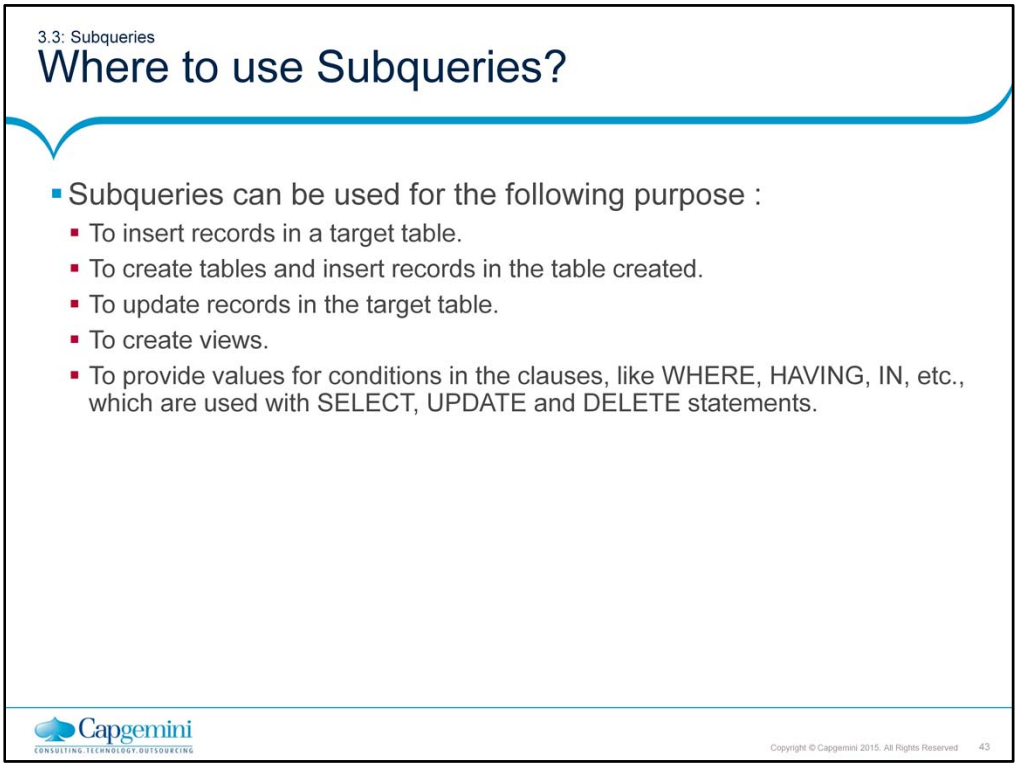
Sub-queries:
As mentioned earlier, since a basic SQL query returns a relation, it can be used to construct composite queries.
Such a SQL query, which is nested within another higher level query, is called a “sub-query”.
This kind of a nested sub-query is useful in cases, where we need to select rows 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.



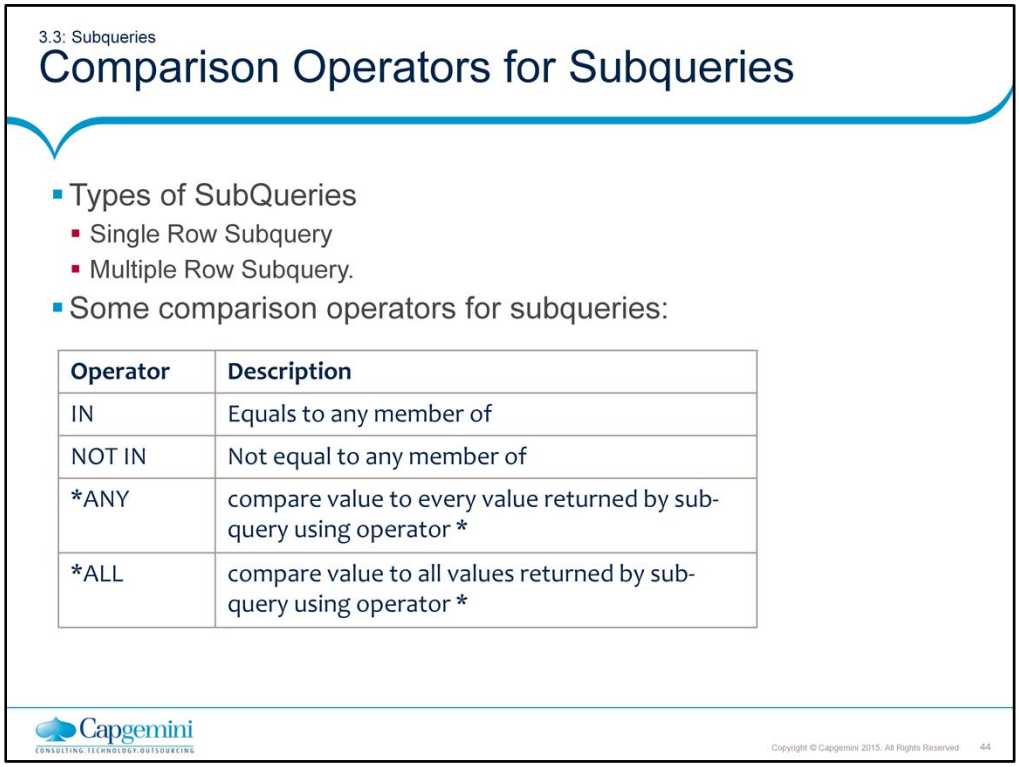
The problem is resolved using a one query inside another. The inner query is called as a subquery or nested query. The subquery result is used by the outer query. The subqueries can have more levels of nesting. The innermost query will execute first. The subquery execute only once before the outer query



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.

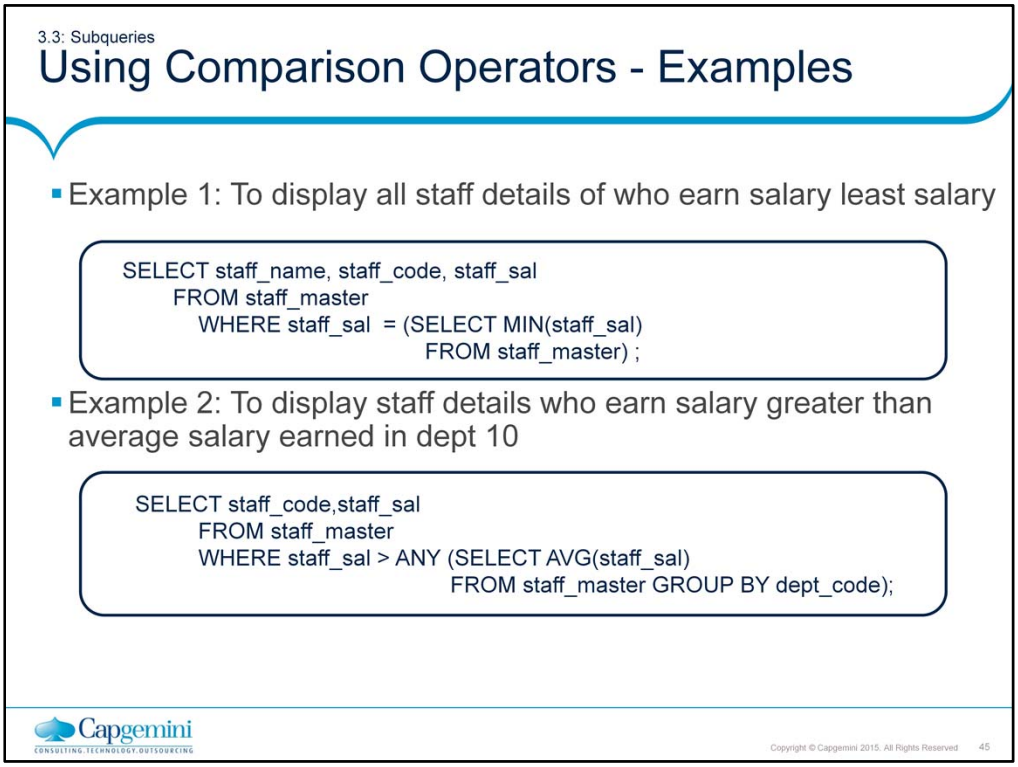


Sub-queries by using Comparison operators:
Sub-queries are divided as “single row” and “multiple row” sub-queries. While single row comparison operators (=, >, >=, <, <=, <>) can only be used in single row sub-queries, multiple row sub-queries can use IN, ANY or ALL.
For example: The assignment operator (=) 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:

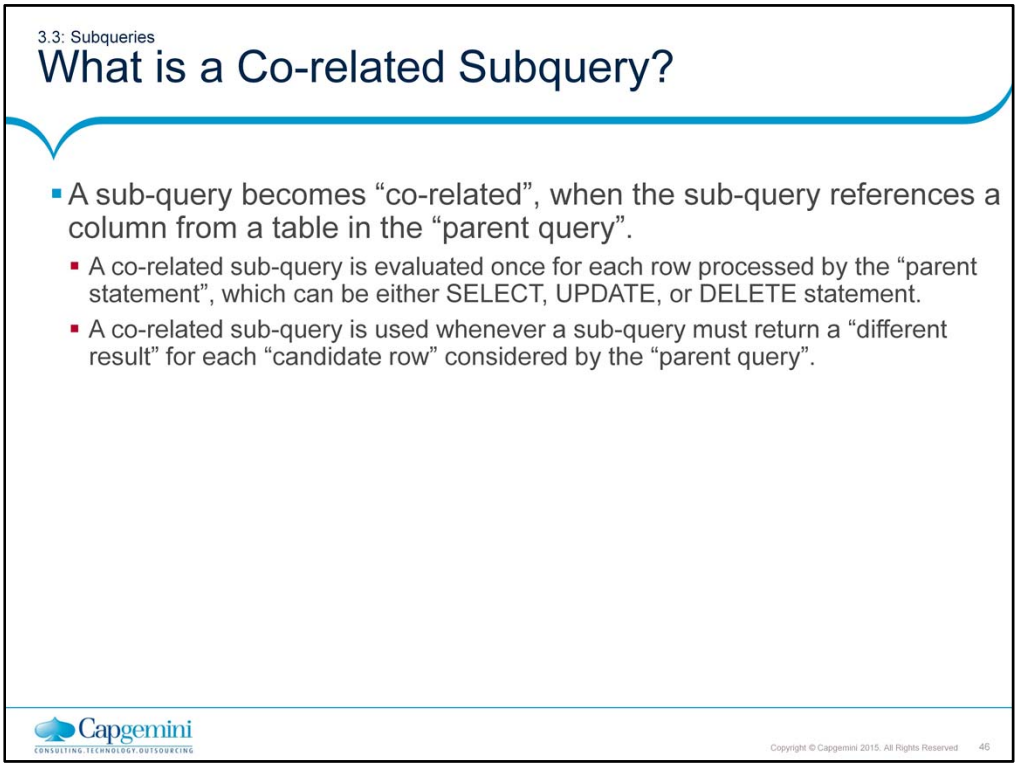
True if the comparison is true for every value of the list of values.
True if sub-query gives a null set (No values)
False if the comparison is false for one or more of the list of values generated by the sub-query.

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.

3.3: Subqueries

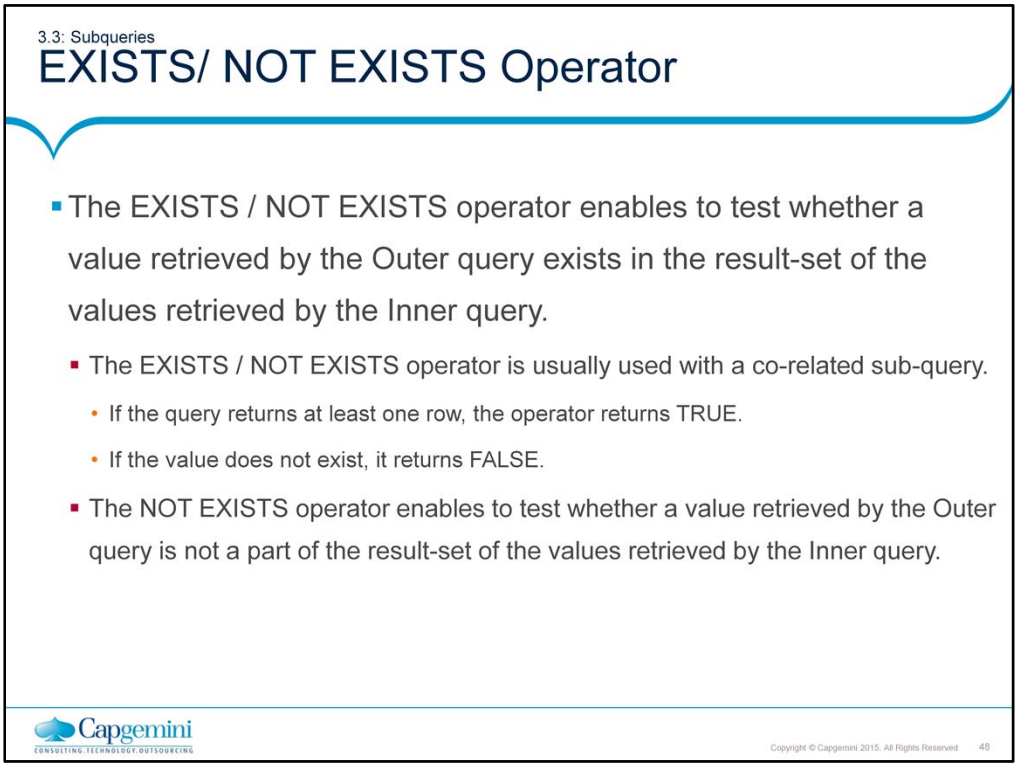
Co-related Subquery -Examples

- 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

 47



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.

3.3: Subqueries

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

 49

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

 50

Quick Guidelines

- If Single row operators are used for a sub query that returns multiple rows, Oracle would throw an error
- If one of the value returned by the sub query is NULL, then the whole sub query returns NULL.
- Correlate sub queries hamper performance , use them only when absolutely required

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 51

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

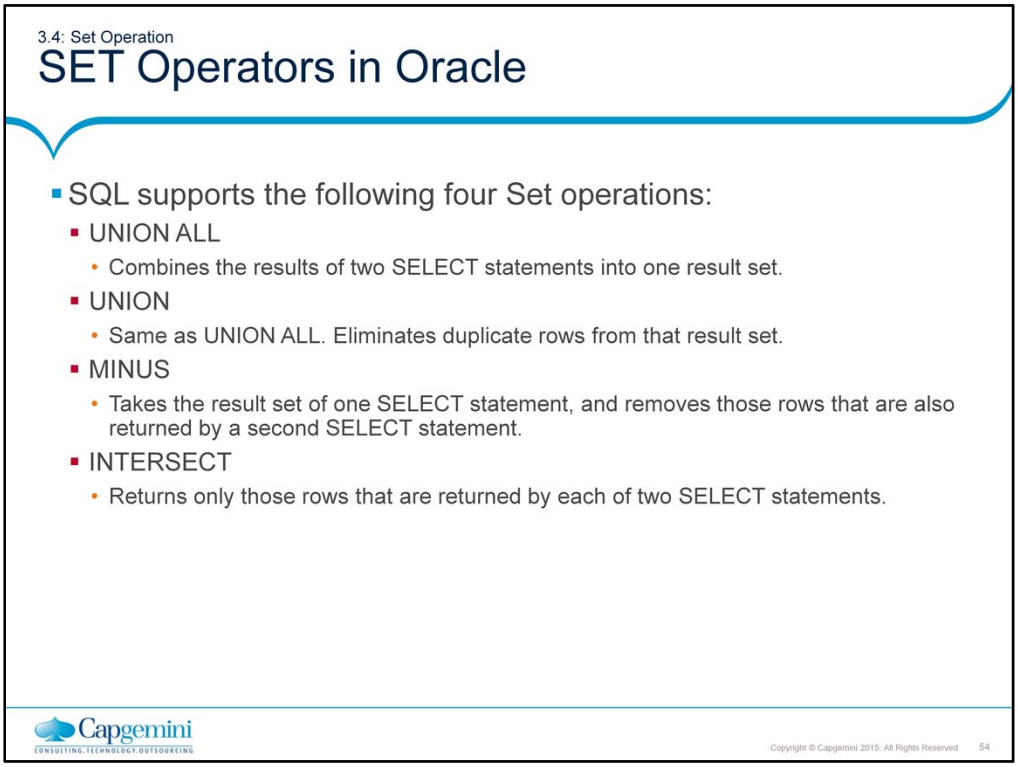
 52

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

 53



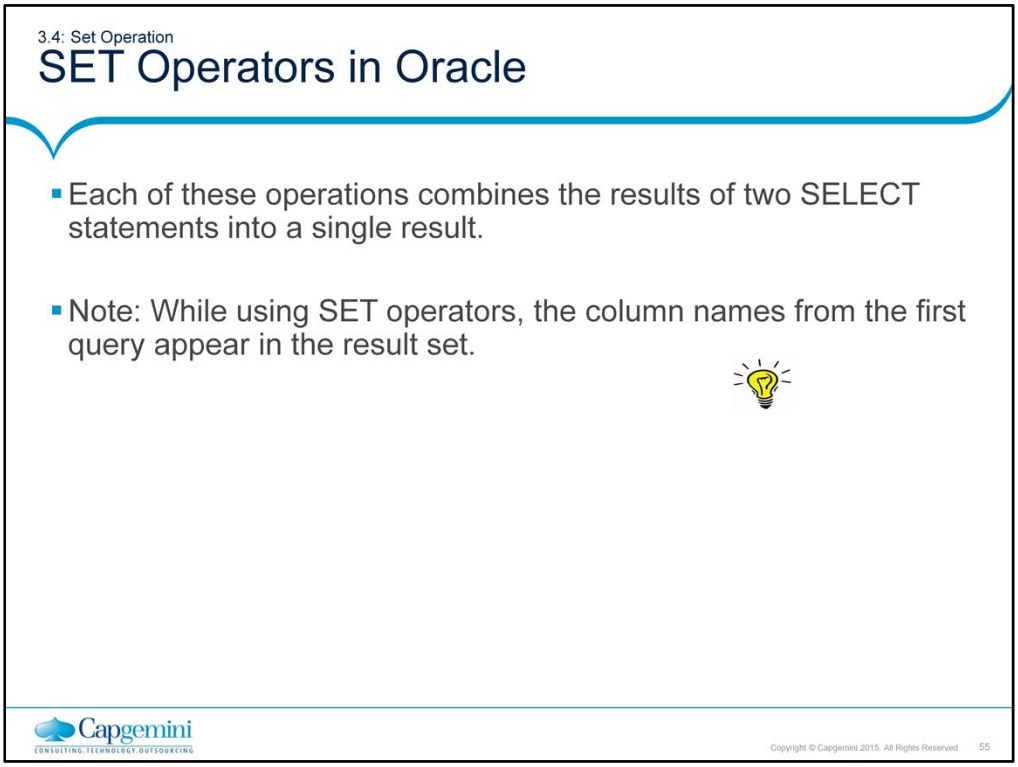
Set Operators:
There are situations when we need to “combine the results” from two or more SELECT statements. SQL enables us to handle these requirements by using “Set operations”.
The result of each SELECT statement can be treated as a set. SQL set operations can be applied on these sets to arrive at a final result.
SQL supports the following four Set operations:

- UNION ALL
- UNION
- MINUS
- INTERSECT

All set operators have equal precedence.

If a SQL statement contains multiple set operators, Oracle evaluates them from the left to right if there are no parentheses explicitly specifying another order.

contd.



SQL statements containing the Set operators are referred to as “compound queries”. Each SELECT statement in a compound query is referred to as a “component query”.

Two SELECT statements can be combined into a compound query by a set operation only if they satisfy the following two conditions:

The “result sets” of both the queries must have the “same number of columns”.

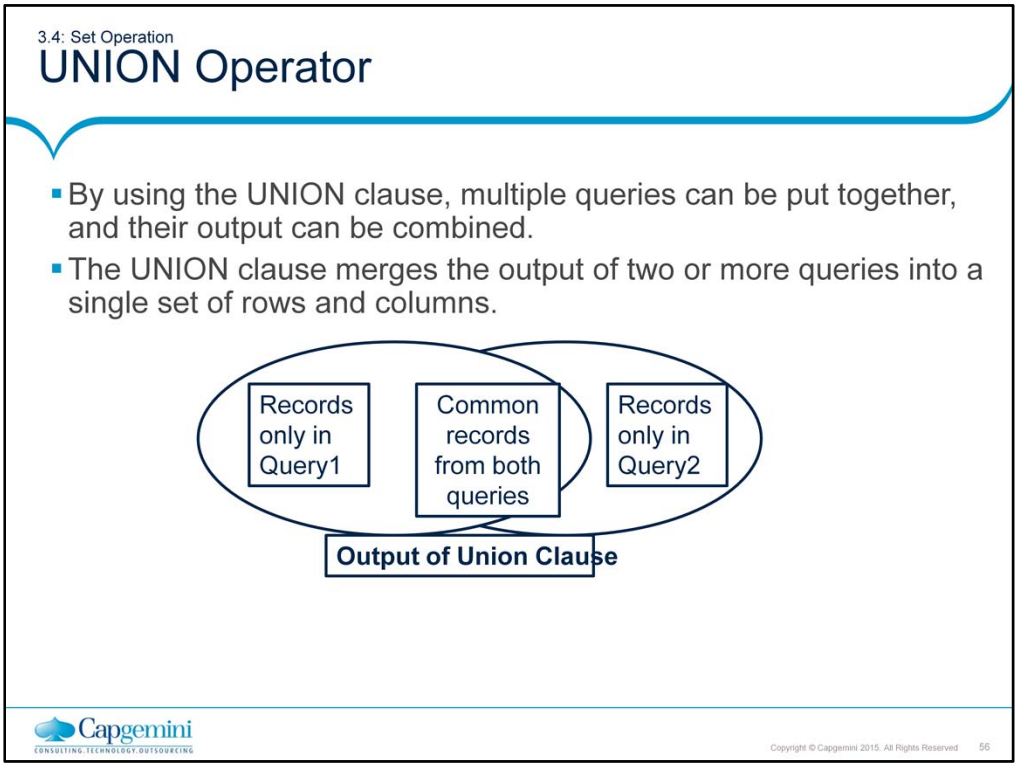
The “datatype” of each column in the “second result set” must match the “datatype” of its corresponding column in the “first result set”.

For example: If component queries select character data, then the datatype of the return values are determined as follows:

If both queries select values of datatype CHAR, then the returned values have datatype CHAR.

If either or both of the queries select values of datatype VARCHAR2, then the returned values have datatype VARCHAR2.

Tip: The datatypes do not need to be the same, if those in the second result set can be automatically converted by Oracle (using implicit casting) to types that are compatible with those in the first result set.



UNION Operator:

The UNION operator returns the records retrieved by either of the queries.

By default, the UNION operator eliminates duplicate records.

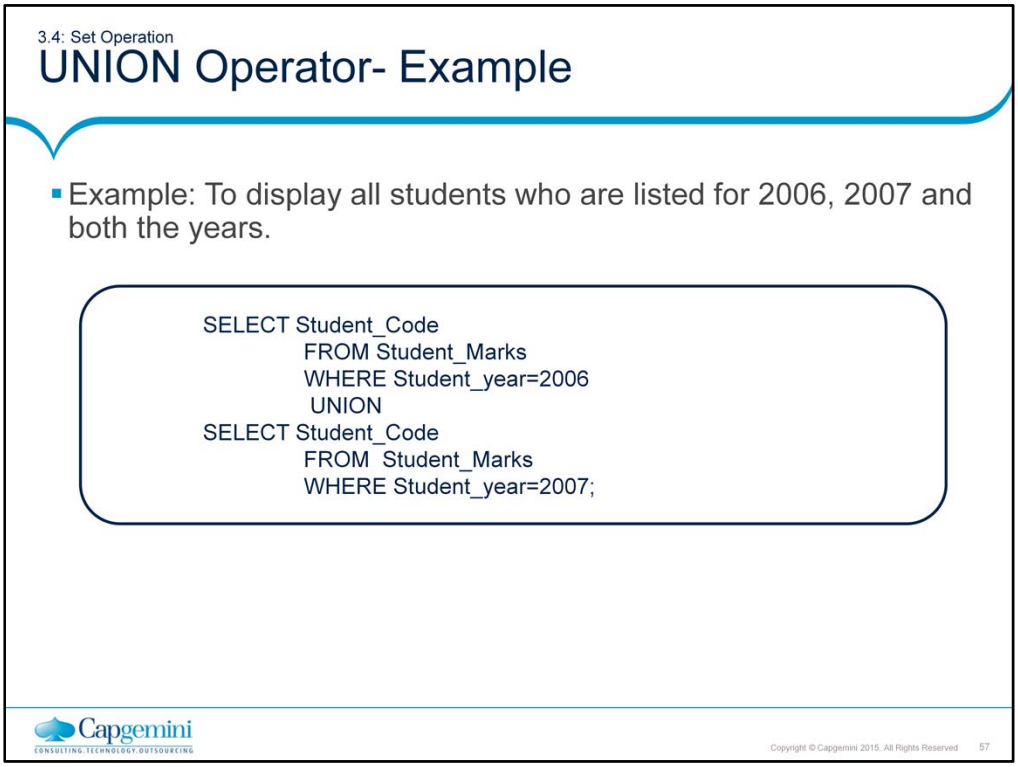
If however we want to retain duplicates, we use UNION ALL instead of UNION.

UNION operates over all of the columns being selected.

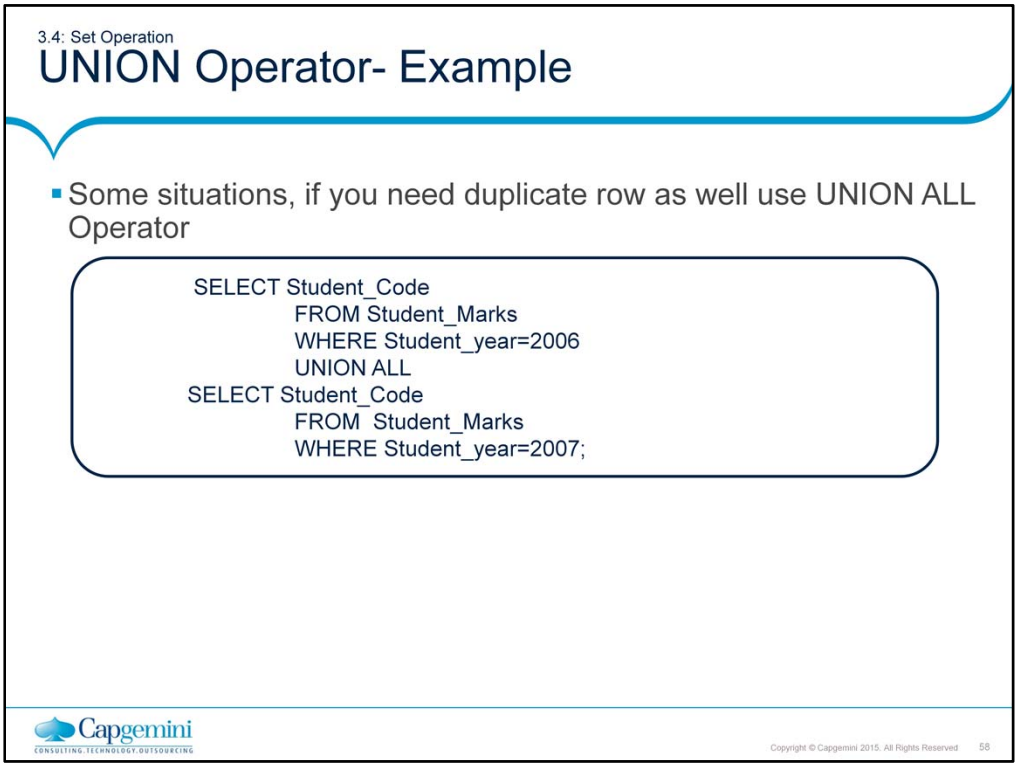
NULL values are not ignored during duplicate checking.

The IN operator has a higher precedence than the UNION operator.

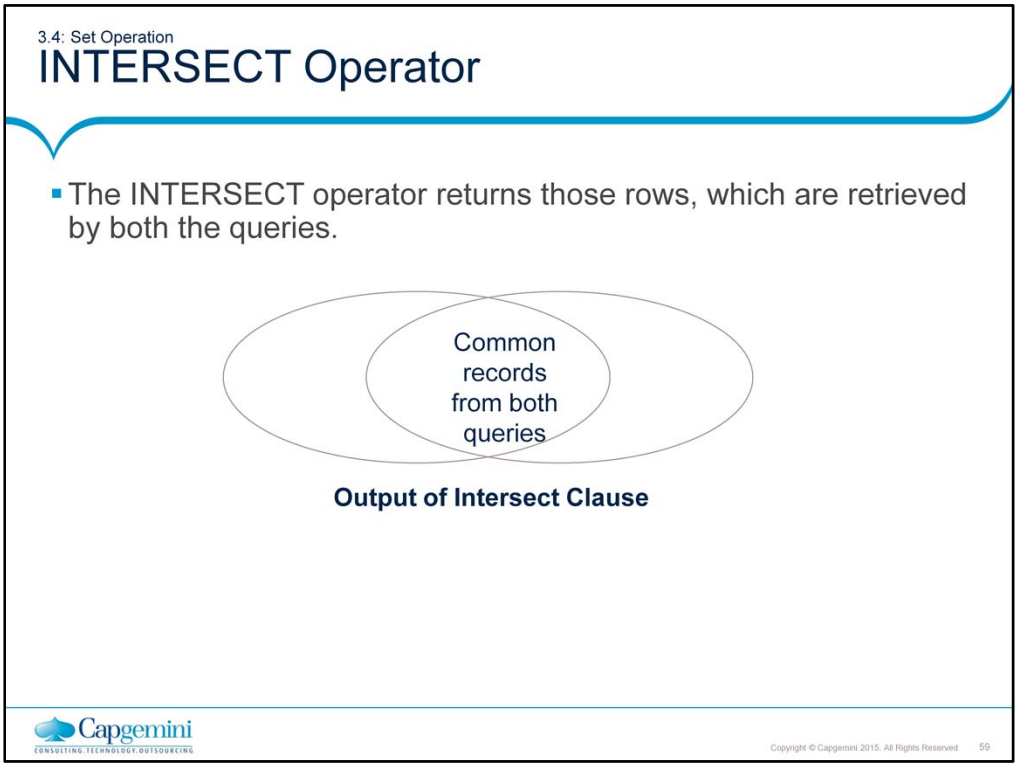
By default, the output is sorted in ascending order of the first column of the SELECT clause.



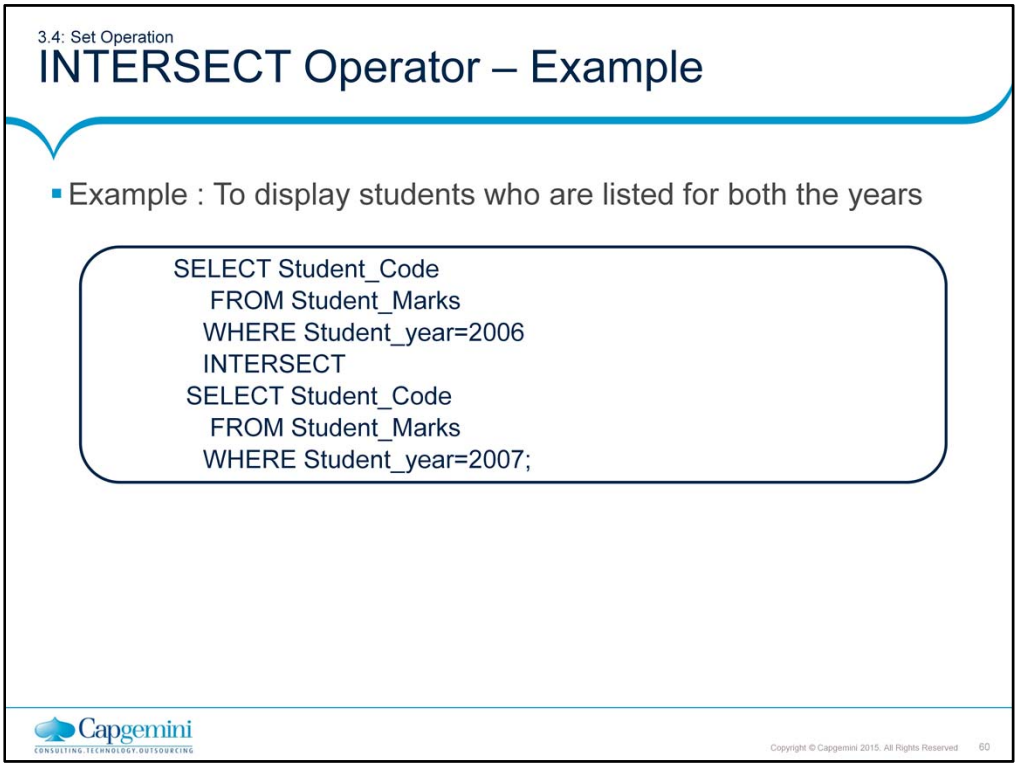
The query on the slide retrieves unique values from first query and second query. It also retrieves common values in both the queries by removing the duplicate records.



The above query since is using UNION ALL will remove results returned by both the queries. The common values will be duplicate.



INTERSECT Operator
INTERSECT result is same on reversing the order
INTERSECT does not ignore NULL values

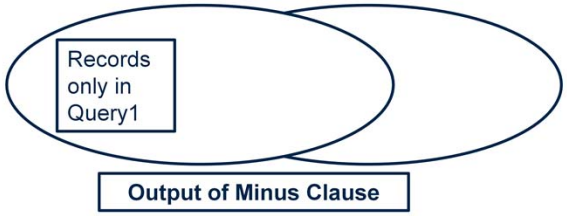


Example of INTERSECT operator:
The example on the slide will only display the common records retrieved by both the queries

3.4: Set Operation

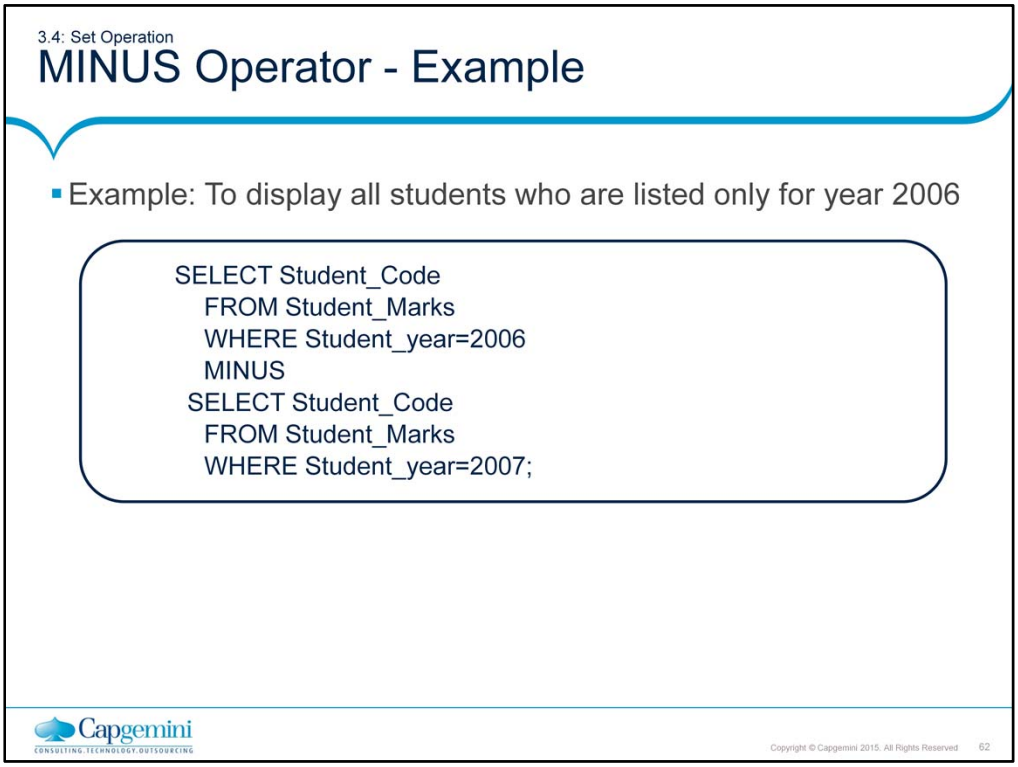
MINUS Operator

- The MINUS operator returns all rows retrieved by the first query but not by the second query.

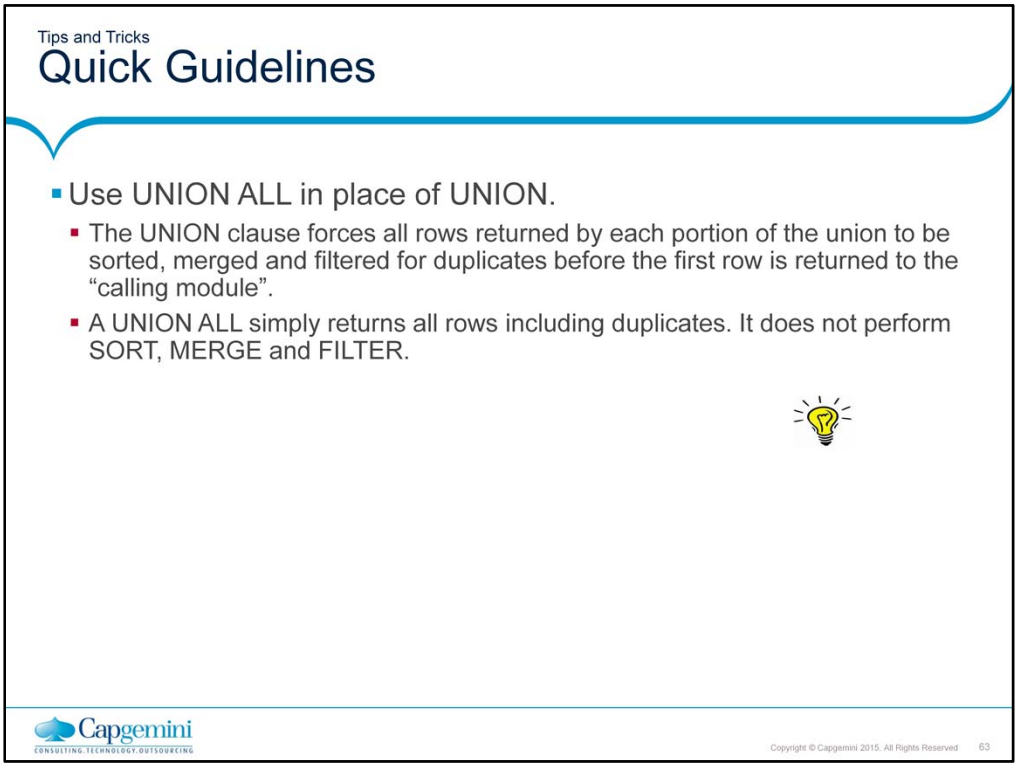


Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 61



Example of MINUS operator:
The query on the slide will show results which are unique to first query



Tips and Tricks:
When using the UNION statement, keep in mind, that the UNION statement performs the equivalent of a SELECT DISTINCT on the final result set, by default. In other words, UNION takes the results of two like record sets, combines them, and then performs a SELECT DISTINCT in order to eliminate any duplicate rows.

This process occurs even if there are no duplicate records in the final record set.

Hence,

If you know that there are duplicate records, and it creates a problem for your application, then use the UNION statement "to eliminate the duplicate rows".

If you know that there will never be any duplicate rows, or if there are duplicates, and it does not create problems in your application, then you should use the UNION ALL statement instead of the UNION statement.

The advantage of the UNION ALL statement is that is does not perform the SELECT DISTINCT function. This saves a lot of server resources from being unnecessarily used.

Summary

- In this lesson you have learnt,
  - Aggregate (Group functions)
    - GROUP BY clause
    - HAVING clause
  - Joins
    - Oracle Proprietary Joins
    - SQL: 1999 Compliant Joins
  - Types of joins
  - Sub-queries
  - Use of Set Operations

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 64

Review – Questions

- Question 1: The AVG function ignores NULL values in the column.
  - True / False
- Question 2: A sub-query can be used for creating and inserting records.
  - True / False
- Question 3: The Set operation that will show all the rows from both the resultsets including duplicates is ____.
  - Option 1: Union All
  - Option 2: Union
  - Option 3: Intersect
  - Option 4: Minus

Review – Questions

- Question 4: The Intersect operator returns ___.
- Question 5: The output of set operators shows the columns names from ___.