SELECT column\_name

FROM Table\_name;

SELECT \*   
FROM Table\_name;

SELECT column\_1 , column\_2

FROM table\_name;

SELECT column\_1\*2 AS multiplay   
FROM table\_name;  
  
***this will show a column named multiply and in each row the values will be multiplied by 2***

SELECT column\_1 "Ailes"   
FROM table\_name;  
  
DESC table\_name;  
  
***this will describe the table column names , data .. etc***  
  
SELECT 'this Id=' || id || ' has price of' || price "newPrice"

FROM games\_of\_the\_year;  
  
***the output here will be one column and in each row (this Id=ID has price of PRICE) and the column name will be newPrice***  
  
SELECT DISTINCT COLUMN\_ID  
  
***this will remove the duplicates from the selected column ,or combination of columns duplicates maybe found in either of the columns but not in both***

FROM table\_name;  
  
SELECT first\_name, last\_name

FROM employee

WHERE last\_name='Taylor';  
  
SELECT \*   
  
FROM games\_of\_the\_year

WHERE price BETWEEN 10 AND 1200;

***this will show all games with prices between 10 and 1200***

SELECT \*

FROM games\_of\_the\_year

WHERE id IN (4,1,5) AND genre IN ('tps','bbs','gcs');   
  
***this will show all the games that have id of 4 or 1 or 5 and that have genre of tps or bbs or gcs***  
  
SELECT last\_name   
  
FROM employees

WHERE last\_name LIKE '\_n%';  
  
***this will show last names that start with one character followed by 'n' letter and a string of letters afterwards***

***% : represents a set of chars***

***\_ : represents single character***

rnfi , anfbo , snlim , zn will be shown on the screen  
  
WHERE job\_id LIKE ' %\\_R% ' ESCAPE ' \ ';

***this means that the \ (it can be any character) is the ESCAPE letter and the \_ is from the original search word***   
  
  
SELECT last\_name , manager\_name  
  
FROM employees

WHERE manager\_name IS NULL;   
  
***this will show all last names of employees that have no managers assigned to .***  
  
SELECT last\_name , manager\_name  
  
FROM employees  
  
WHERE manager\_name IS NOT NULL;  
  
  
***this will only show all last names of employees that have managers assigned to .***  
  
SELECT last\_name, hire\_date, department\_id

FROM employees

ORDER BY department\_id NULLS FIRST;   
  
***this will sort data by department\_id in ascending fashion where the NULL department\_id's will be at first.***  
  
***by default null is last in ascending order and first in descinding order***

SELECT last\_name, hire\_date, department\_id

FROM employees

ORDER BY department\_id DESC NULLS LAST;   
  
***this will sort data by department\_id in descending fashion where the NULL department\_id's will be at last.***

SELECT last\_name, hire\_date

FROM employees

ORDER BY department\_id ;   
  
***(we can order using a column that is not included in the select clause )***

SELECT last\_name, hire\_date

FROM employees

ORDER BY department\_id , last\_name ;   
  
***(we can order by more than one column )***

DUAL TABLES

SELECT (319/29)+12

FROM DUAL ;   
  
***(the result will be shown in a table that have on column called (319/29)+12 and one raw that have the result 23 )***  
  
**---------------string functions**  
  
SELECT last\_name

FROM employees

WHERE LOWER(last\_name)='laem' ;  
  
***this will compare last\_names in lowercase with the word 'laem' and will show only last names that their lowercase is 'laem'***  
  
SELECT last\_name

FROM employees

WHERE UPPER(last\_name)='LOL' ;  
  
***this will compare last\_names in uppercase with the word 'LOL' and will show only last names that their uppercase is 'LOL'***  
  
SELECT last\_name

FROM employees

WHERE INITCAP(last\_name)='Liemmfis' ;  
  
***INITCAP() function sets the first letter of each word in uppercase, and all other letters in lowercase.***  
  
***so in this example it will compare the word 'Liemmfis' with all the last\_names by changing the last\_names first letter to uppercase letter and the rest of letters to lowercase letters***  
  
SELECT CONCAT(first\_name,last\_name)

FROM employee;  
  
***CONCAT() will combine two strings together , in this case it will combine the first names and the last names and it will show the two columns combined into one .***   
  
SELECT SUBSTR(last\_name , 1 , 3)   
  
FROM employees ;  
  
***this will show only 3 chars of the last\_name starting from position 1 (the first char)   
  
if we only declare the starting position without determining the length the string, it will start from the position we declared and the rest of it will be shown***  
  
SELECT LENGTH(last\_name)

FROM employees ;

***this will show the column of last\_name but with the lengths of the strings***  
  
SELECT last\_name , INSTR(last\_name , 'a' , 1 , 2 )   
  
FROM employees;  
  
***this will show the position a in each last\_name starting from the first position and showing the position of the second occurrence***SELECT LPAD(last\_name,10,'\*')  
  
FROM employees;

***the first argument is the string we want to pad and the second argument is the number of chars in the padded string and the last argument is the character to pad with***

***\*\*\*\*dave here the padding is 8***

***\*\*\*\*\*car here the padding is also 8  
  
right padding is the same but from right to left***  
  
SELECT TRIM (LEADING 'a' FROM last\_name)

FROM employee;

***this will trim the first a's in the last\_name***

SELECT TRIM (TRAILING 'a' FROM last\_name)

FROM employee;   
  
***this will trim the last a's in the last\_name***  
SELECT TRIM (BOTH 'a' FROM last\_name)

FROM employee;   
  
***this will trim both the first and the last a's in the last\_name***   
  
SELECT REPLACE(last\_name,'a','\*')

FROM employees ;

***this will change the letters a in each last name with \****

***NOte: if we dont add the argument '\*' then all the a's in the last name will be remomved***   
  
SELECT first\_name , last\_name , salaray , department\_id

FROM employees

WHERE department\_id=:enter\_dept\_id;  
  
***this will make oracle ask you to enter the department\_id every time you run the sql code***  
  
---------------numurical functions   
  
ROUND(45.926) = 46

ROUND(45.926,0)=46

ROUND (45.926 , 2) =45.93

ROUND (45.926 , -1) = 50

TRUNC (45.926 , 2 ) =45.92

TRUNC ( 45.926 , 0 ) = 45

TRUNC ( 45.926) = 45  
  
SELECT first\_name , last\_name , salaray , department\_id

FROM employees

WHERE MOD(department\_id , 2)=0;  
  
***this will show first\_name , last\_name , salary , department\_id of employees that have even deparment\_id numbers***  
  
---------------dates functions

SYSDATE ***shows the current system date***

SELECT last\_name , hire\_date

FROM employees

WHERE MONTHS\_BETWEEN (SYSDATE , hire\_date) >240

***MONTHIS\_BETWEEN returns number of months between two dates and returns negative if the first date is earlier than the second date***   
  
SELECT ADD\_MONTHS (hire\_date , 12)

AS "NEXT YEAR"   
  
FROM employee;  
  
***this will add 12 months to the hire date and if the number is negative it will subtract 12 months***   
  
SELECT NEXT\_DAY (hire\_date , 'saturday')

AS "Next Saturday"

FROM employee;

***this will show the date of the next Saturday after the hire\_date***  
  
SELECT LAST\_DAY (hire\_date)

AS "last day in the month"

FROM employee;

***this will show the date of the last day in the month of the hire\_date***  
  
SELECT hire\_date , ROUND(hire\_date , 'Month')

FROM employee;   
  
***this will round the month of the date , if day > , 15 month will round up else it will round down***  
  
  
SELECT hire\_date , ROUND(hire\_date , 'Year')

FROM employee;   
  
***this will round the month of the date , if month > 6 , year will round up else it will round down***.  
  
SELECT hire\_date , TRUNC(hire\_date , 'month')

FROM employee;   
  
***this is going to start the month from the first day***

SELECT hire\_date , TRUNC(hire\_date , 'year')

FROM employee;

***this is going to start the year from the first day and the first month***

TO\_CHAR (date column name , 'format model you sepcify')

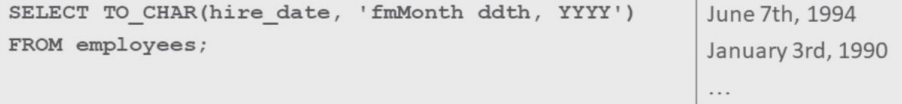
***It is used to convert date format to character data using a desired format model***

We can use these with format model the following :   
  
sp : to spell out a number

Th : to have the number appear as (1st , 2nd ,3rd ) ordinal .

Fm : to remove the padded blanks or leading zeros from the out put

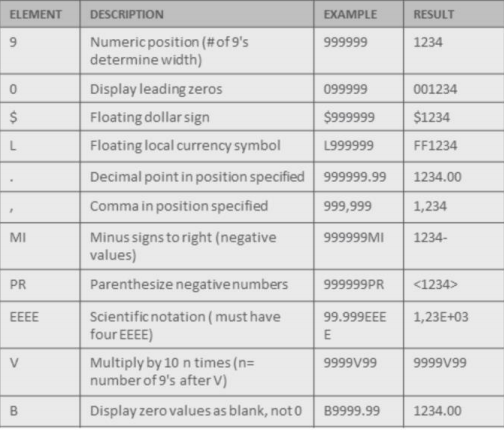
For example :





TO\_CHAR (number , 'format model you sepcify')

***It is used to convert number format to character data using a desired format model***



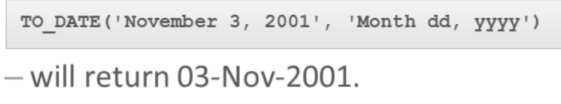
TO\_NUMBER (character string , 'format model you sepcify')

***It is used to convert character format to number data using a desired format model***

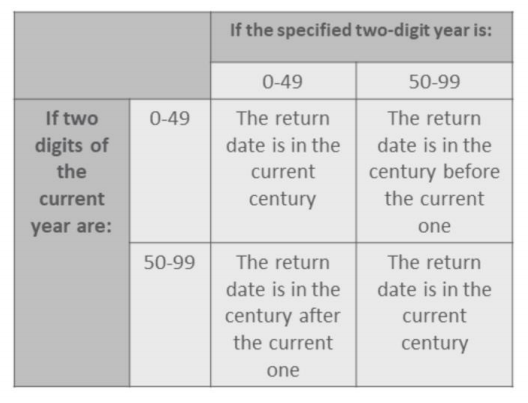


TO\_DATE (character string , 'format model you sepcify')

***It is used to convert character format to Date data using a desired format model***



***The RR rules table :***



NVL (column that may contain a null value , the substitution value )

***It is used to substitute a null value in a certain column to a specified value we choose***

NVL2(column that may contain a null value , substitution 1 if the value is not null , substitution 2 if the value is NULL )   
  
***it returns expression 1 if the value is not null and returns expression 2 if the value is null***

NULLIF (expression 1 , expression 2 )

***If expression 1 equals expression 2 it returns NULL***

***Else***

***It returns expression 1***

COALESCE ( expression 1 , expression 2 , …. Expression n )

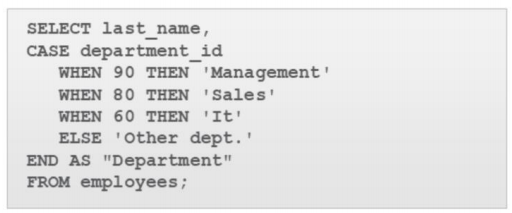
***If the first expression is null the function continues down the line until a not null expression is found the returns that expression***

CASE expr when comparison\_expr1 THEN return\_expr1.

when comparison\_expr2 THEN return\_expr2.

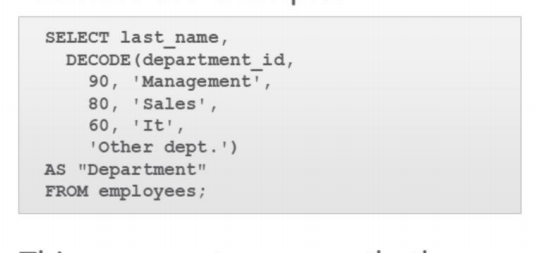
when comparison\_expr3 THEN return\_expr3.

ELSE else\_expr

END

DECODE (column1 , search 1 , result 1

[ , search 2 , result 2 , …, ]

 [, default] )

**Natural join :**

select name , genre,copy\_price , mv\_id , revenue

from table\_1 natural join table\_2

This join will return columns from the table\_1 and their related revenue from the table\_2 based on the common column mv\_id

Note : it is optional to write the common column name in the select clause

**Cross join :**

select name, genre,copy\_price ,revenue

from table\_1 cross join table\_2

similar to natural join there must be a common column between the two tables.

The result set represents all possible row combinations from the two tables.

**Using join :**

**•**In a natural join, if the tables have columns with the same names but different data types, the join causes an error.

•To avoid this situation, the join clause can be modified with a USING clause.

•The USING clause specifies the columns that should be used for the join.

•The columns referenced in the USING clause should not have a qualifier (table name or alias) anywhere in the SQL statement.

•The USING clause allows us to use WHERE to restrict rows from one or both tables

select mv\_id, name , genre , copy\_price , rating , release\_date , revenue

from table\_1 join table\_2 using(mv\_id)

where revenue=’400$’ here revenue is from the table\_2

**on join :**

•What if the columns to be joined have different names, or if the join uses non-equality comparison operators such as <, >, or BETWEEN ?

•We can't use USING, so instead we use an ON clause.

•This allows a greater variety of join conditions to be specified.

•The ON clause also allows us to use WHERE to restrict rows from one or both tables.

select mv\_id,name,genre,copy\_price,rating,release\_date,revenue

from table\_1 join table\_3

on (table\_1.mv\_id=table\_3.revenue\_id) #here we enforce the common columns to have the same names

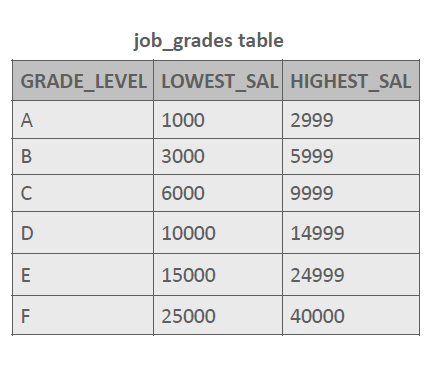
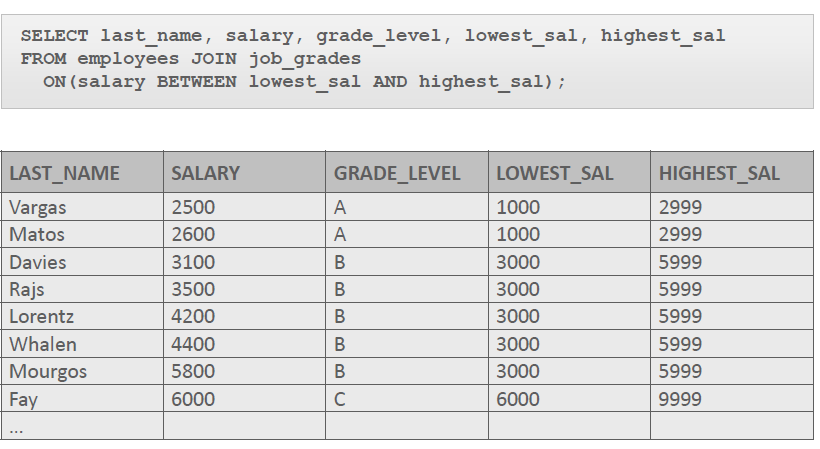
order by mv\_id

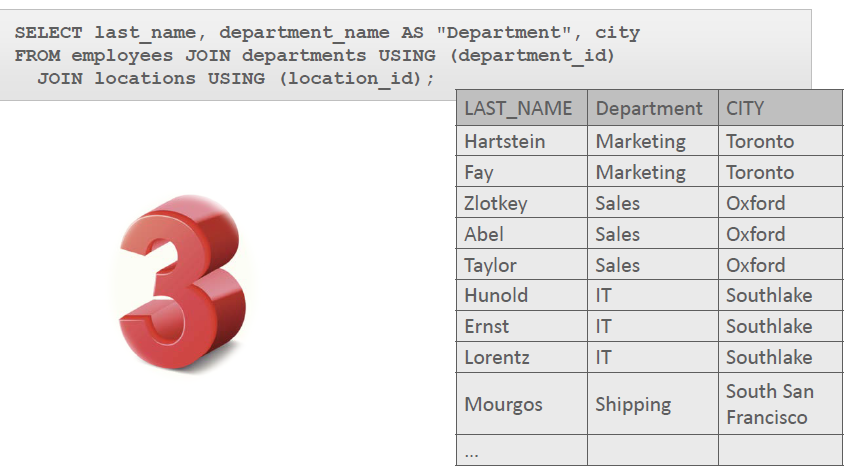
•Sometimes you may need to retrieve data from a table that has no corresponding column in another table.

•Suppose we want to know the grade\_levelfor each employees salary.

•The job\_gradestable does not have a common column with the employees table.

•Using an ON clause allows us to join the two tables



**Joining Three Tables**

**Inner and outer joins :**

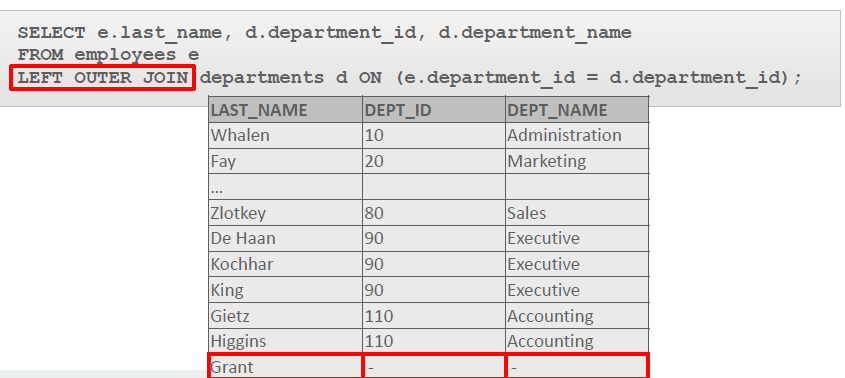
•a join of two or more tables that returns only the matched rows is called an inner join.

•When a join returns the unmatched rows as well as the matched rows, it is called an outer join.

•Outer join syntax uses the terms "left, full, and right".

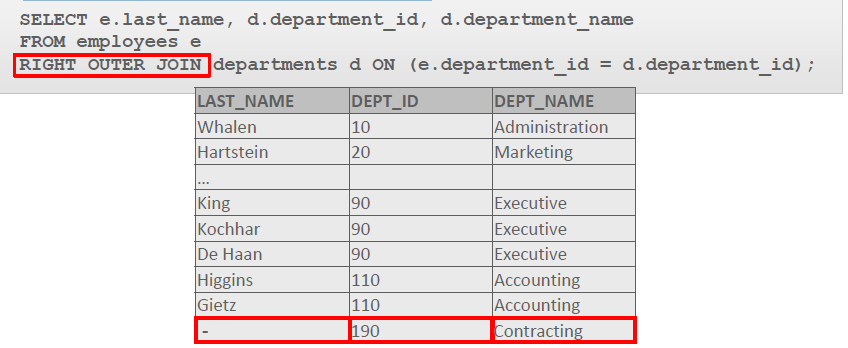
**Left outer join :**

This query will return all employee last names, both those that are assigned to a department and those that are not.



**Right outer join :**

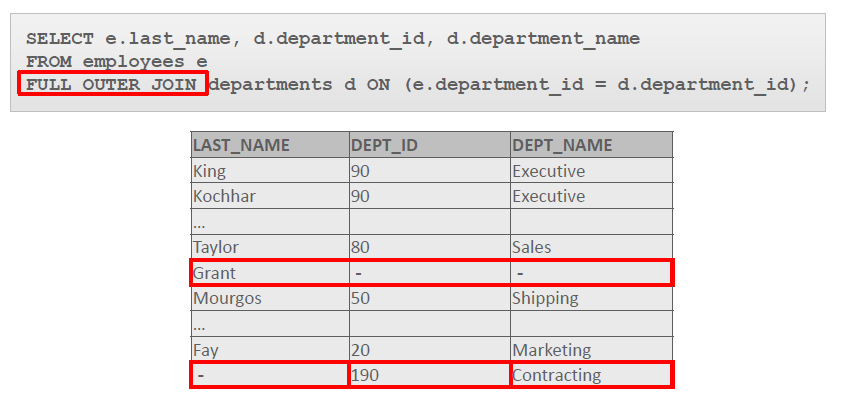
This right outer join would return all department IDs and department names, both those that have employees assigned to them and those that do not.



**Full outer join :**

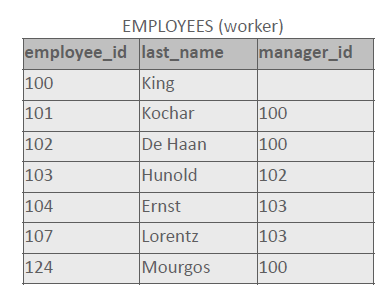
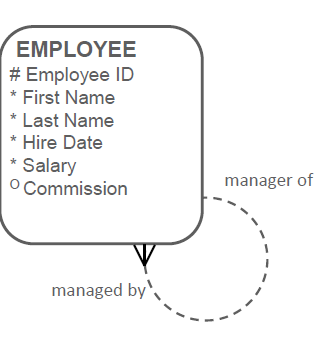
•It is possible to create a join condition to retrieve all matching rows and all unmatched rows from both tables.

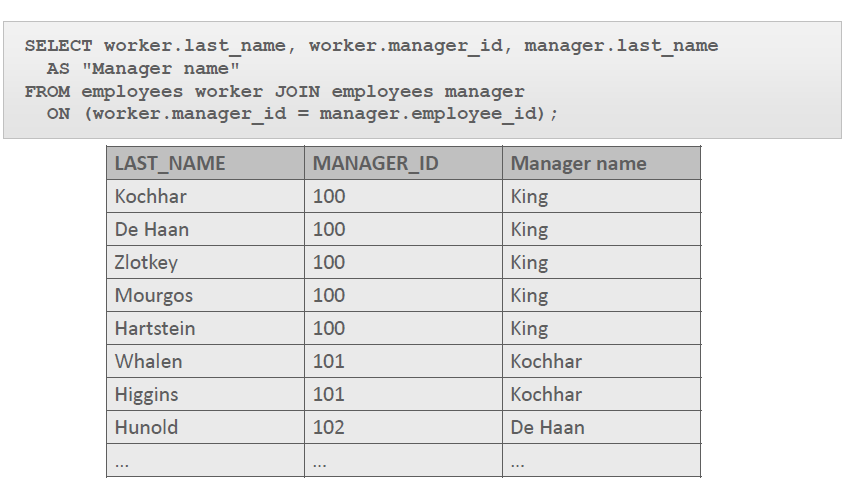
•Using a full outer join solves this problem.

•The result set of a full outer join includes all rows from a left outer join and all rows from a right outer join combined together without duplication.

**Self Join :**

•Once we have a real employees table, a special kind of join called a self-join is required to access this data.

•A self-join is use to join a table to itself as if it was two tables.



**Group functions :**

•Each function returns one result.

**MIN**:

Used with columns that store any data type to return the minimum value.

select min(rating) as "min"

from table\_1

this will return a single value .

it can be used with strings , numbers , and dates

**Max**:

Used with columns that store any data type to return the maximum value.

select max(rating) as "max"

from table\_1

this will return a single value .

it can be used with strings , numbers , and dates

**Sum :**

Used with columns that store numeric data to find the total or sum of values.

select sum(rating) as "sum"

from table\_1

this will return a single value .

it can be used with numbers only .

**AVG**:

Used with columns that store numeric data to compute the average.

select avg(rating) as "average"

from table\_1

this will return a single value .

it can be used with numbers only .

select avg(nvl(column\_name , 0)) can be used to calculate the average of the column and including the null values by replacing them with zeros

**count :**

Returns the number of elements in a column , note : null values are not counted.

select count(rating)

from table\_1

it can be used with strings , numbers , and dates

•

**COUNT(\*)**

returns the number of rows in a table.

select count(\*)

from table\_1

note : count(\*) includes the nulls and the duplicates

**VARIANCE and STDDEV :** are used to calculate the variance and the standard deviation

it can be used with numbers only .

**Note** :

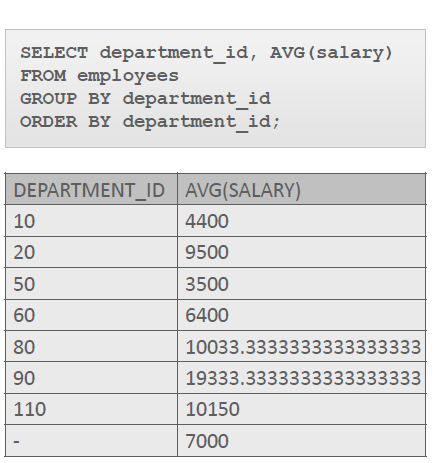
•Group functions ignore null values.

•Group functions cannot be used in the WHERE clause.

•MIN, MAX and COUNT can be used with any data type; SUM, AVG, STDDEV, and VARIANCE can be used only with numeric data types.

**Group by :**

•You use the GROUP BY clause to divide the rows in a table into smaller groups.

•You can then use the group functions to return summary information for each group.

Notes : Group functions require that any column listed in the SELECT clause that is not part of a group function must be listed in a GROUP BY clause.

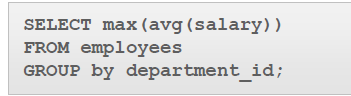
–If you include a group function (AVG, SUM, COUNT, MAX, MIN, STDDEV, VARIANCE) in a SELECT clause along with any other individual columns, each individual column must also appear in the GROUP BY clause.

–You cannot use a column alias in the GROUP BY clause.

–The WHERE clause excludes rows before they are divided into groups.

**Nesting Group Functions**

Group functions can be nested to a depth of two when GROUP BY is used.



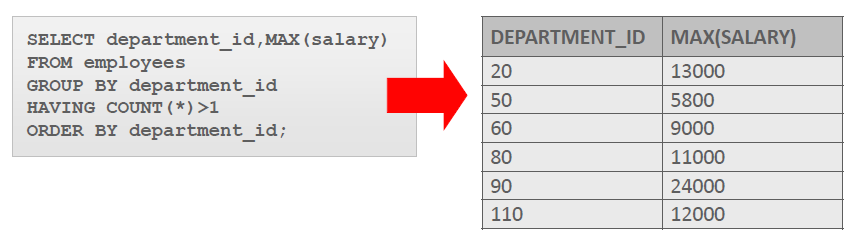
the query will find the average salary for each department, and then from that list, select the single largest value.

**Having ;**

**•**In the same way you used the WHERE clause to restrict the rows that you selected, you can use the HAVING clause to restrict groups.

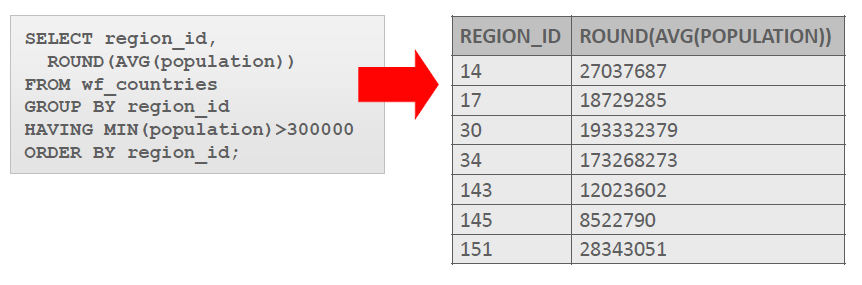
•In a query using a GROUP BY and HAVING clause, the rows are first grouped, group functions are applied, and then only those groups matching the HAVING clause are displayed.

Example 1 :



Example 2 :

•This query finds the average population of the countries in each region.

•It then only returns the region groups with a lowest population greater than three hundred thousand.

**Single row Subquery and single column:**

•placing one query inside the other query.

•The inner query is called a "subquery."

•Subqueries can be placed in a number of SQL clauses, including the WHERE clause, the HAVING clause, and the FROM clause.

•The subquery is enclosed in parentheses.

•The subquery is placed on the right side of the comparison condition.

•A subquery cannot have its own ORDER BY clause.

•The two types of subqueries are:

–Single-row subqueries that use single-row operators (>, =, >=, <, <>, <=) and return only one row from the inner query.

–Multiple-row subqueries that use multiple-row operators (IN, ANY, ALL) and return more than one row from the inner query.

select \*

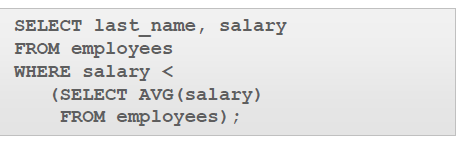
from table\_1

where rating<(

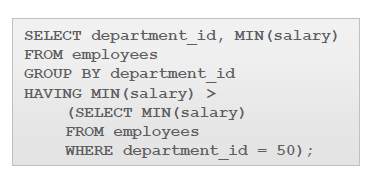
select rating

from table\_1

where name='jhon wick')

example 2 :

Example 3 :

•Which departments have a lowest salary that is greater than the lowest salary in department 50?

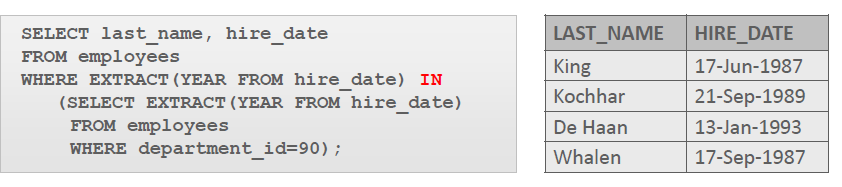
**Multiple row Subquery and single column :**

•The subquery will need to return several rows.

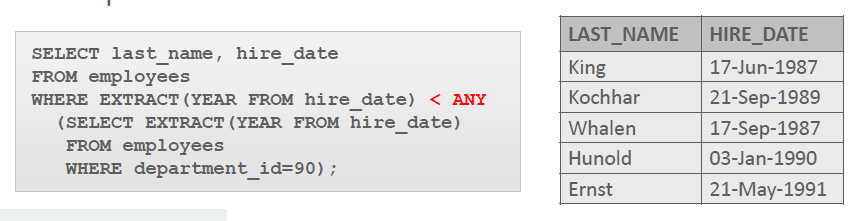
•We achieve this using multiple-row subqueries and the three comparison operators: IN, ANY, and ALL.

**IN :**

•For example, we are interested in all the employees that were hired the same year as an employee in department 90.

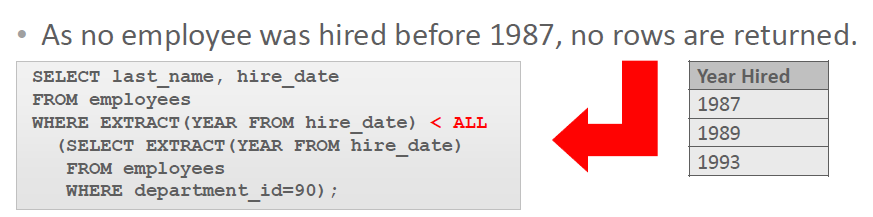


**ANY :**

•The ANY operator is used when we want the outer-query WHERE clause to select the rows which match the criteria (<, >, =, etc.) of at leastone value in the subquery result set.

**ALL:**

•The ALL operator is used when we want the outer-query WHERE clause to select the rows which match the criteria ( <, >, =, etc.) of allof the values in the subquery result set.



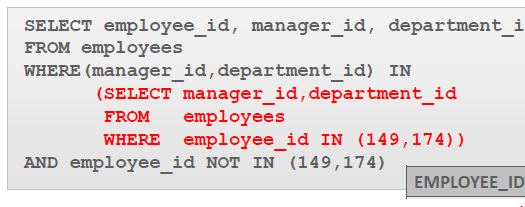
•Suppose that one of the values returned by a multiple-row subquery is null, but other values are not.

•If IN or ANY are used, the outer query will return rows which match the non-null values.

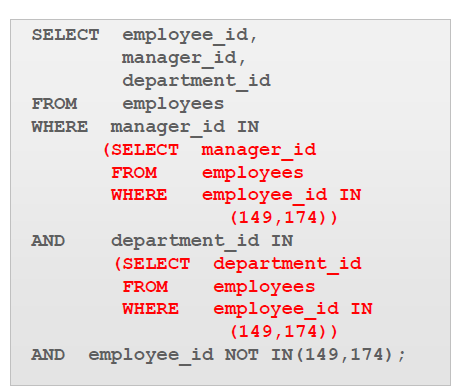
•If ALL is used, the outer query returns no rows because ALL compares the outer query row with every value returned by the subquery, including the null.

**Multiple column Subquery :**

•A multiple-column subquery can be either pair-wise comparisons or non-pair-wise comparisons.

*multiple-column pair-wise subquery:*

*A non-pair-wise multiple-column subquery:*



**EXISTS & NOT EXISTS in Subqueries**

•EXISTS, and its opposite NOT EXISTS, are two clauses that can be used when testing for matches in subqueries.

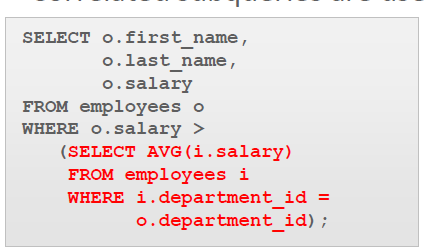
•EXISTS tests for a TRUE, or a matching result in the subquery.

**Correlated Subquery Example:**

In this subquery there is a connection between the subquery and the outer query.

•Whose salary is higher than the average salary of their department?

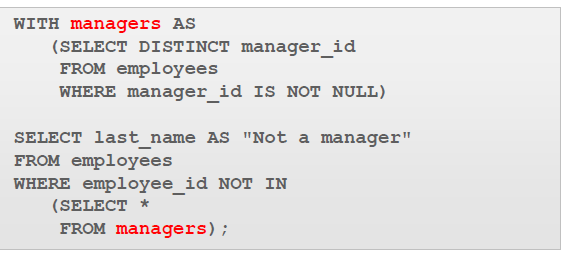
•To answer that question, we need to write a correlated subquery.



**WITH Clause**

**•**If you have to write a very complex query with joins and aggregations used many times, you can write the different parts of the statement as query blocks and then use those same query blocks in a SELECT statement.

•Oracle allows you to write named subqueries in one single statement, as long as you start your statement with the keyword WITH.

•The WITH clause retrieves the results of one or more query blocks and stores those results for the user who runs the query.

Create a Copy of a Table: • The reason for this is that the CREATE TABLE …. AS (SELECT …) statement that is used to create the copy of the table, copies both the rows and the not null constraints, but does not copy the primary key –foreign key constraints.

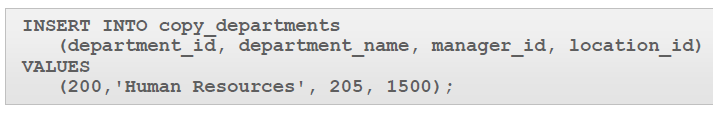


**DDL** stands for Data Definition Language. **DML** stands for Data Manipulation Language. **DDL** statements are used to create database, schema, constraints, users, tables etc. **DML** statement is used to insert, update or delete the records.

**DML :**

**INSERT :**

•The INSERT statement is used to add a new row to a table.



To explicitly add a null value to a column that allows nulls, use the NULL keyword in the VALUES list.

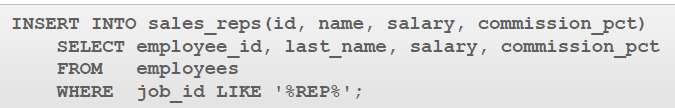
•To specify empty strings and/or missing dates, use empty single quotation marks (with no spaces between them like this '') for the missing data.

•Special values such as SYSDATE and USER can be entered in the VALUES list of an INSERT statement.

•SYSDATE will put the current date and time in a column.

•USER will insert the current session's username, which is OAE\_PUBLIC\_USER in Oracle Application Express.

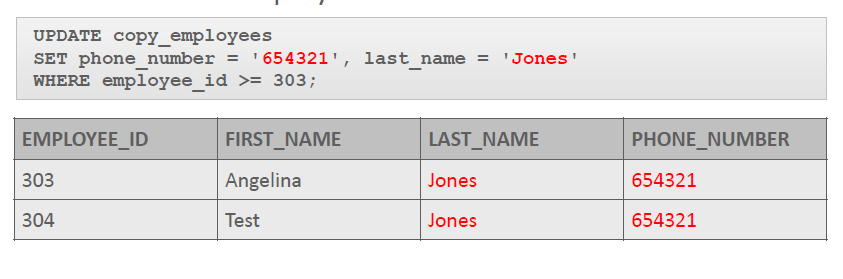
**Using A Subquery To insert Rows :**

•In the example shown, a new table called SALES\_REPS is being populated with copies of some of the rows and columns from the EMPLOYEES table.

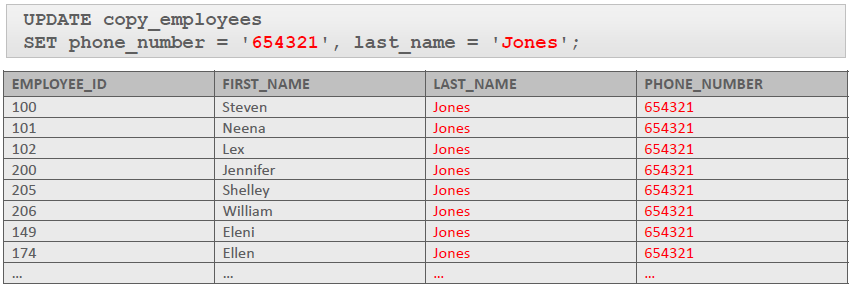
•The subquery is not enclosed in parentheses as is done with the subqueries in the WHERE clause of a SELECT statement.

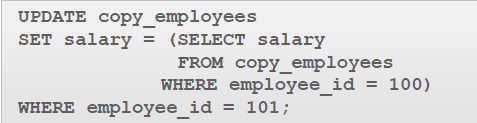
**UPDATE :**

•The UPDATE statement is used to modify existing rows in a table.

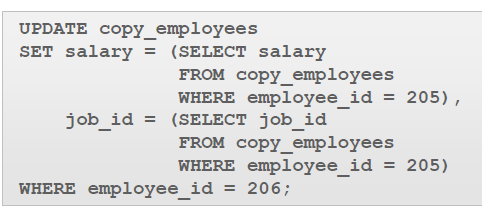


•If the WHERE clause is omitted, every row in the table will be updated.



•We can use the result of a single-row subquery to provide the new value for an updated column.

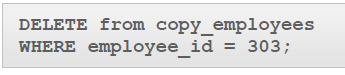
Updating Two Columns with Two Subquery Statements



**DELETE**

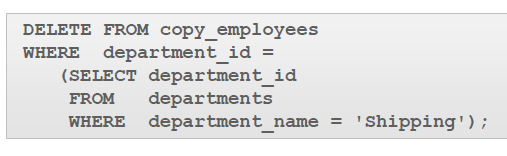
•The DELETE statement is used to remove existing rows in a table.

•The example shown uses the copy employee table to delete one row—the employee with ID number 303.



•All rows in the table are deleted if you omit the WHERE clause.

•Subqueries can also be used in DELETE statements.



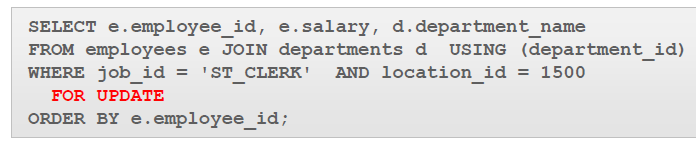
**FOR UPDATE Clause in a SELECT Statement**

•When a SELECT statement is issued against a database table, no locks are issued in the database on the rows returned by the query you are issuing.

•Most of the time this is how you want the database to behave—to keep the number of locks issued to a minimum.

•Sometimes, however, you want to make sure no one else can update or delete the records your query is returning while you are working on those records.

•This is when the FOR UPDATE clause is used.

•As soon as your query is executed, the database will automatically issue exclusive row-level locks on all rows returned by your SELECT statement, which will be held until you issue a COMMIT or ROLLBACK command.

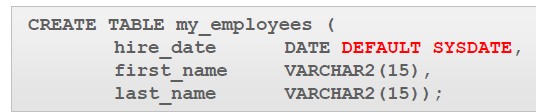
**Default :**

•Each column in a table can have a default value specified for it.

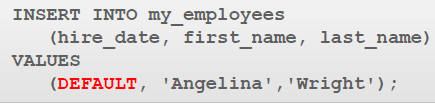
•In the event that a new row is inserted and no value for the column is assigned, the default value will be assigned instead of a null value.

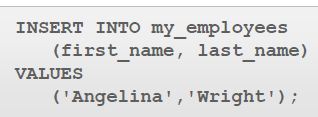
•The default value can be a literal value, an expression, or a SQL function such as SYSDATE and USER, but the value cannot be the name of another column.

•The default value must match the data type of the column.

•DEFAULT can be specified for a column when the table is created or altered.

•The INSERT example using the my\_employeestable shows the explicit use of DEFAULT:



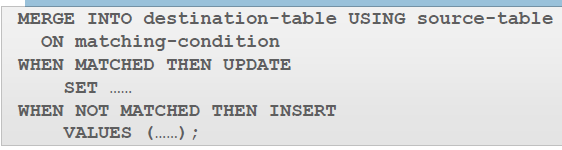
•Implicit use of DEFAULT

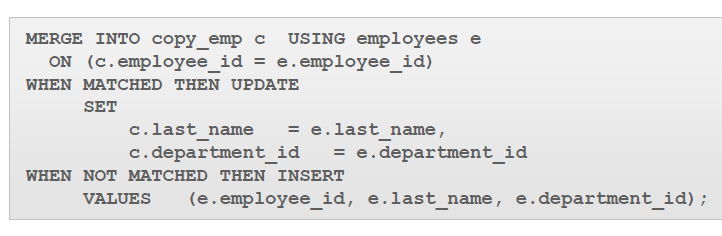
**Merge :**

•Using the MERGE statement accomplishes two tasks at the same time. MERGE will INSERT and UPDATE simultaneously. If a value is missing, a new one is inserted.

•One row at a time is read from the source table and compared to rows in the destination table using the matching condition.

•If a matching row exists in the destination table, the source row is used to update one or more columns in the matching destination row.

•If a matching row does not exist, values from the source row are used to insert a new row into the destination table.



**Multi table inserts :**

•Multi-table inserts can be unconditional or conditional. In an unconditional multi-table insert, Oracle will insert all rows returned by the subquery into all table insert clauses found in the statement.

•In a conditional multi-table insert, you can specify either ALL or FIRST.

•Specifying ALL:

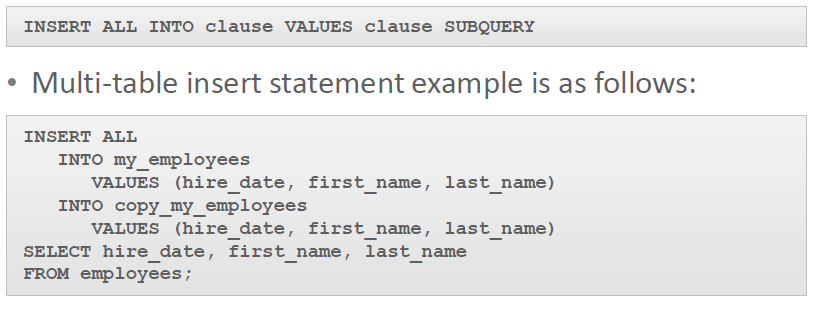
–If you specify ALL, the default value, the database evaluates each WHEN clause regardless of the results of the evaluation of any other WHEN clause.

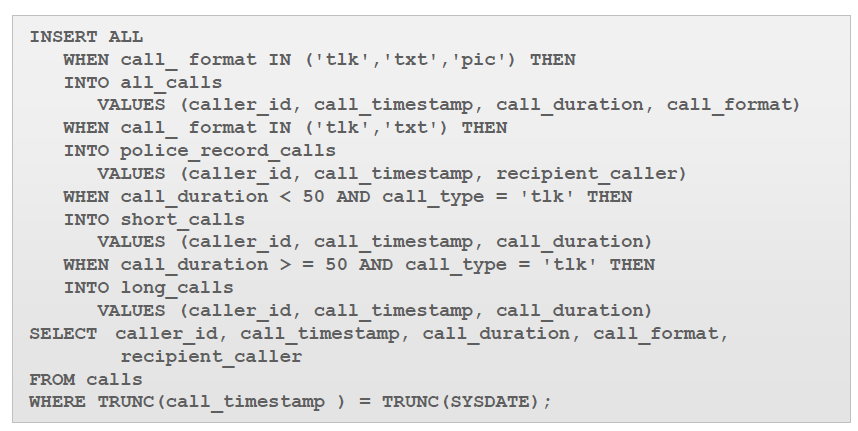
–For each WHEN clause whose condition evaluates to true, the database executes the corresponding INTO clause list.

•Specifying FIRST:

–If you specify FIRST, the database evaluates each WHEN clause in the order in which it appears in the statement.

–For the first WHEN clause that evaluates to true, the database executes the corresponding INTO clause and skips subsequent WHEN clauses for the given row.





**Database Schema Objects**

•The main database object types are:

–Table

–Index

–Constraint

–View

–Sequence

–Synonym

•Some of these object types can exist independently and others can not.

•The database stores the definitions of all database objects in the Data Dictionary, and these definitions are accessible to all users of the database as well as to the database itself.

**Table name rules :**

–Must begin with a letter

–Must be 1 to 30 characters long

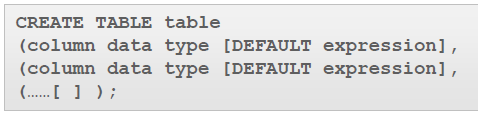
–Must contain only A -Z, a -z, 0 -9, \_ (underscore), $, and #

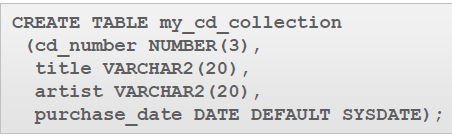
–Must not duplicate the name of another object owned by the same user

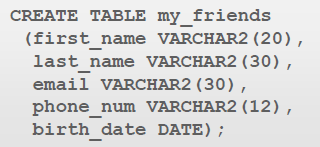
–Must not be an Oracle Server reserved word

•Table names are not case sensitive.

**Create table :**







**Common data types :**

•The most commonly used column data types for character and number values are below.

•For character values:

–CHAR (fixed size, maximum 2000 characters)

–VARCHAR2 (variable size, maximum 4000 characters)

–CLOB (variable size, maximum 128 terabytes)

•For number values:

–NUMBER (va•

The most commonly used column data types for date, time, and binary values are below.

•For date and time values:

–DATE

–TIMESTAMP ….

–INTERVAL

•For binary values (eg. multimedia: JPG, WAV, MP3, and so on):

–RAW (variable size, maximum 2000 bytes)

–BLOB (variable size, maximum 128 terabytes)riable size, maximum precision 38 digits)

•The TIMESTAMP data type is an extension of the DATE data type which allows fractions of a second.

•For example, TIMESTAMP(3) allows 3 digits after the whole seconds, allowing values down to milliseconds to be stored.

ALTER :

•ALTER TABLE statements are used to:

–Add a new column , or drop a column

–Modify an existing column

–Define a DEFAULT value for a column

–•To add a new column, use the SQL syntax shown:Drop a column



•Modifying a column can include changes to a column's data type, size, and DEFAULT value.

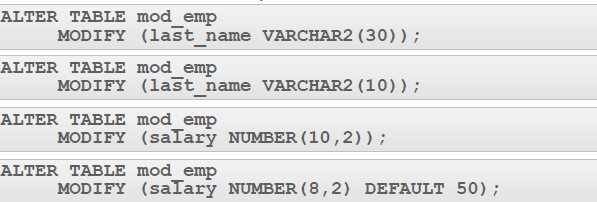
•Rules and restrictions when modifying a column are:

–You can increase the width or precision of a numeric column.

–You can increase the width of a character column.

–You can decrease the width of a NUMBER column if the column contains only null values, or if the table has no rows.

–For VARCHAR types, you can decrease the width down to the largest value contained in the column.

•A change to the DEFAULT value of a column affects only later insertions to the table.

**Dropping a column**

–A column containing data may be dropped.

–Only one column can be dropped at a time.

–You can't drop all of the columns in a table; at least one column must remain.

–Once a column is dropped, the data values in it cannot be recovered.



**Set unused columns :**

•Dropping a column from a large table can take a long time.

•A quicker alternative is to mark the column as unusable.

•The column values remain in the database but cannot be accessed in any way, so the effect is the same as dropping the column.

•In fact, you could add a new column to the database with the same name as the unused column.

•The unused columns are there, but invisible!

**Drop table :**

•When a DROP TABLE statement is issued:

–All data is deleted from the table.

–The table's description is removed from the Data Dictionary.

•Only the creator of the table or a user with DROP ANY TABLE privilege (usually only the DBA) can remove a table.

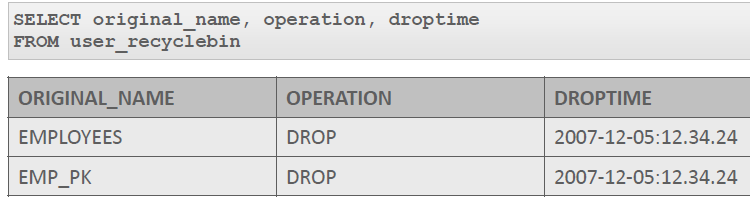


**Flashback table :**

•If you drop a table by mistake, you may be able to bring that table and its data back.



To show the user recycle bin :



**RENAME table :**



**Truncate table :**

•Truncating a table removes all rows from a table and releases the storage space used by that table.

•When using the TRUNCATE TABLE statement:

–You cannot roll back row removal.

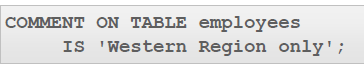
–You must be the owner of the table or have been given DROP ANY TABLE system privileges.

•The DELETE statement also removes rows from a table, but it does not release storage space.

•TRUNCATE is faster than DELETE because it does not generate rollback information.



**Comment on a table :**

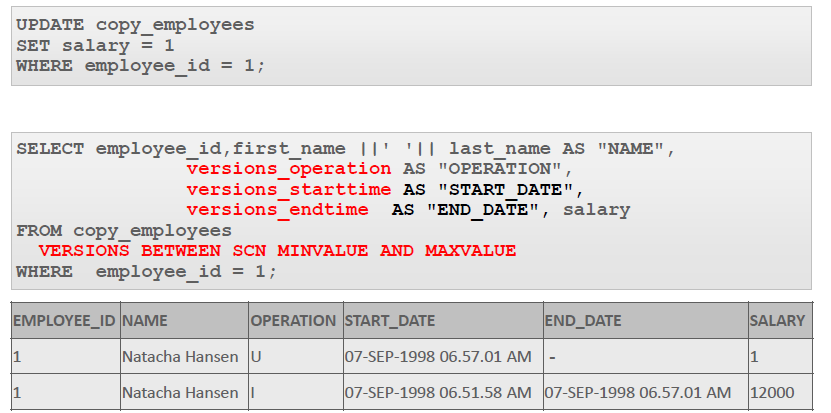




To remove the comment :



**Flash back query :**



**Five Types of Constraints**

–NOT NULL constraints , will also appear in the data dictionary as a CHECK constraint

–UNIQUE constraints , U

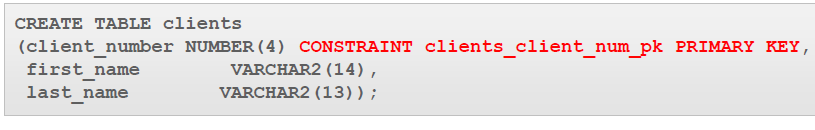
–PRIMARY KEY constraints ,P

–FOREIGN KEY constraints , R

–CHECK constraints , C

**Constraints at the Column Level :**

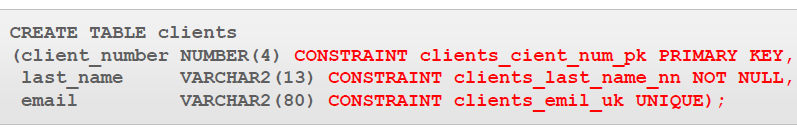
•To establish a column-level constraint, the constraint must be defined in the CREATE TABLE statement as part of the column definition.

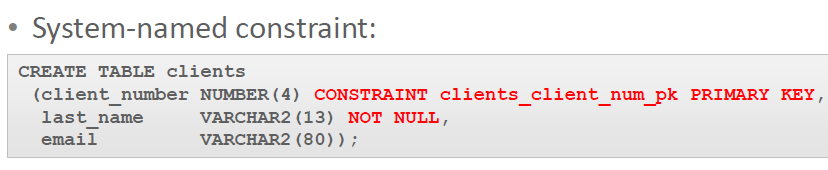


•The name of the constraint is clients\_client\_num\_pk.

•It enforces the business rule that the client\_numberis the primary key of the clients table.

•A naming convention can be the combination of the tablenameabbreviated and a column name abbreviated followed by the constraint abbreviation: table-name\_column-name\_constraint-type





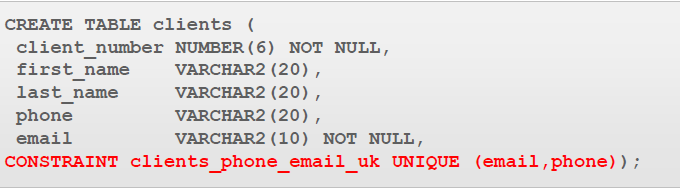
•Two constraints have been created:

–A user-named constraint named clients\_client\_num\_pk, to enforce the rule that client\_numberis the primary key

–A system-named constraint named SYS\_Cn(where n is a unique integer) to enforce the rule that last\_namescannot be null.

**Constraints at the Table Level**

•In the example shown, the unique constraint is listed last in the CREATE TABLE statement.



**Basic Rules for Constraints**

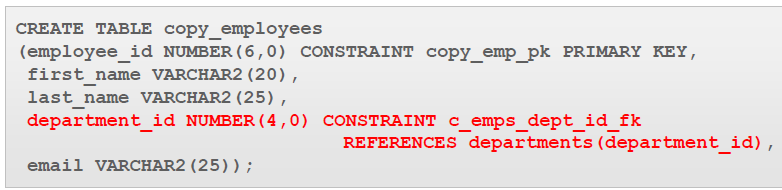
•Constraints that refer to more than one column (a composite key) must be defined at the table level

•The NOT NULL constraint can be specified only at the column level, not the table level

•UNIQUE, PRIMARY KEY, FOREIGN KEY, and CHECK constraints can be defined at either the column or table level

•If the word CONSTRAINT is used in a CREATE TABLE statement, you must give the constraint a name

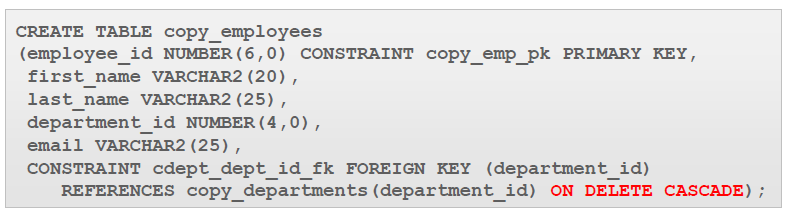
**FOREIGN KEY Constraint Syntax**

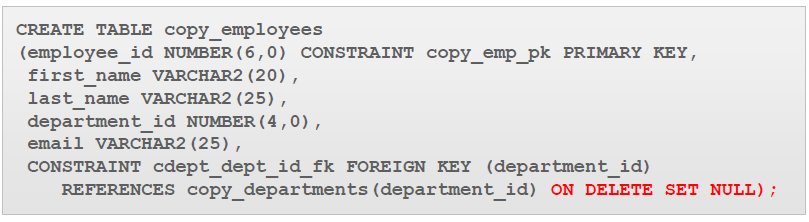


**ON DELETE CASCADE**

• when defining a foreign key in the child table we need to add on delete cascade so that in the future we can delete rows from the referenced table(parent table) without having errors , because a foreign key cannon exist without the primary key that is being referenced.

**Note** : when a row in the parent table is deleted it will be deleted too in the child table if we use on delete cascade

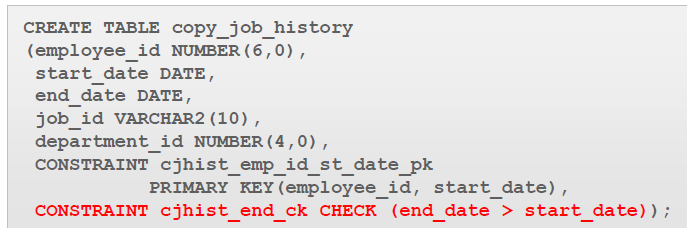


•Rather than having the rows in the child table deleted when using an ON DELETE CASCADE option, the child rows can be filled with null values using the ON DELETE SET NULL option.

**CHECK Constraints**

•The CHECK constraint explicitly defines a condition that must be met.

•This CHECK constraint ensures that a value entered for end\_dateis later than start\_date.

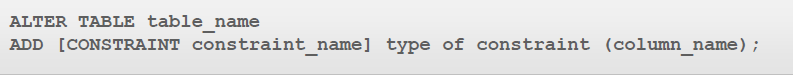
•As this CHECK CONSTRAINT is referencing two columns in the table, it MUST be defined at table level.

•There is no limit to the number of   
CHECK constraints that you can   
define on a column.

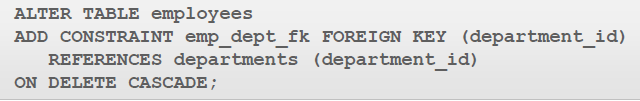
**Managing Constraints**

•The ALTER TABLE statement is used to make changes to constraints in existing tables.

•These changes can include adding or dropping constraints, enabling or disabling constraints, and adding a NOT NULL constraint to a column.

•To add a constraint to an existing table, use the following SQL

•If the constraint is a FOREIGN KEY constraint, the REFERENCES keyword must be included in the statement.

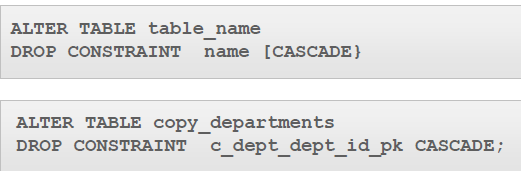


•NOT NULL constraints can be added only if the table is empty or if the column contains a value for every row , we use modify with not null because it is already null so we are modifying it :

**Dropping Constraints**

•To drop a constraint, you need to know the name of the constraint.

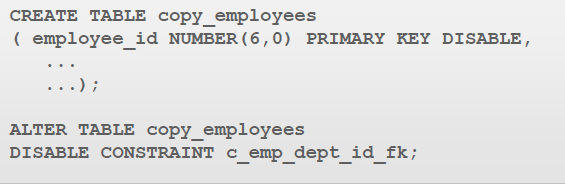
•The CASCADE option of the DROP clause causes any dependent constraints also to be dropped.



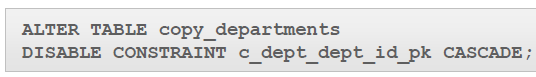
**Disabling Constraints**

•You can disable a constraint without dropping it or re-creating it by using the ALTER TABLE option DISABLE.

•You can use the DISABLE clause in both the ALTER TABLE statement and the CREATE TABLE statement.



•The CASCADE clause disables dependent integrity constraints. If the constraint is later enabled, the dependent constraints are not automatically enabled.



**Enabling Constraints**

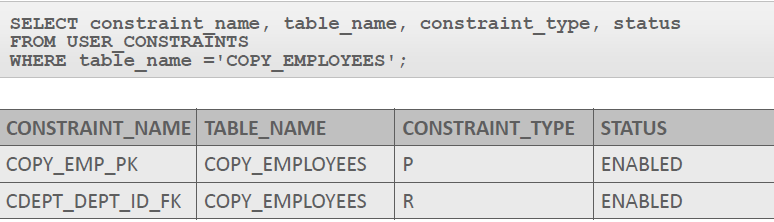
Same syntax as disabling constraints .

•If you enable a constraint, that constraint applies to all the data in the table.

•All the data in the table must fit the constraint.

•If you enable a UNIQUE KEY or PRIMARY KEY constraint, a UNIQUE or PRIMARY KEY index is created automatically.

•Enabling a PRIMARY KEY constraint that was disabled with the CASCADE option does not enable any foreign keys that are dependent on the primary key.

**Viewing Constraints**

**View :**

•A view, like a table, is a database object.

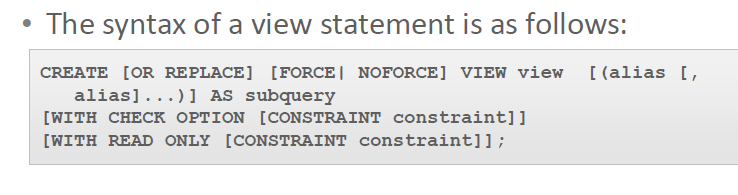
•However, views are not "real" tables.

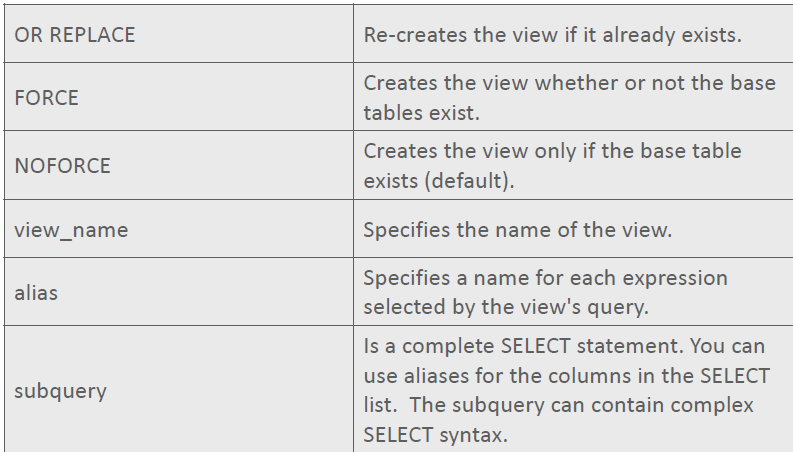
•They are logical representations of existing tables or of another view.

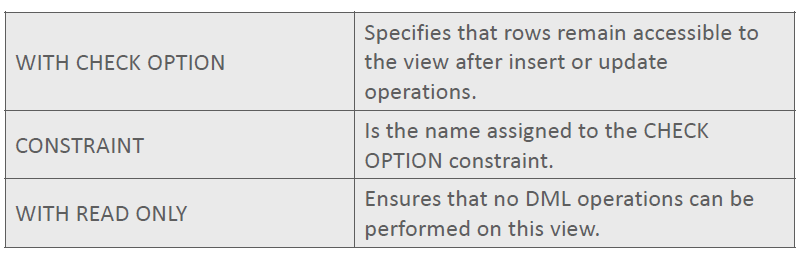
•Views contain no data of their own.

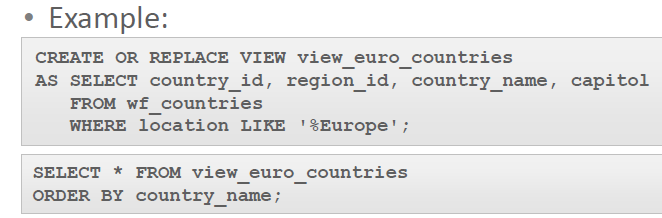
•They function as a window through which data from tables can be viewed or changed.

•The tables on which a view is based are called "base" tables.

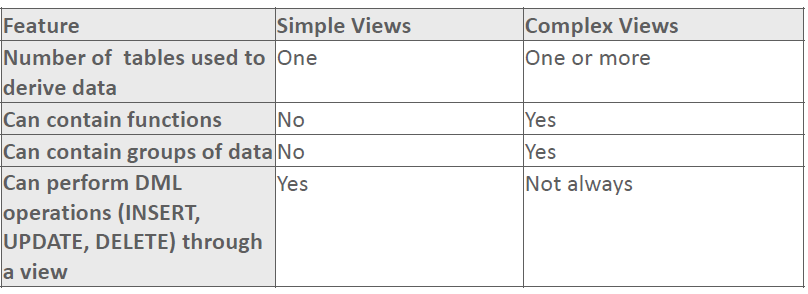
•The view is a query stored as a SELECT statement in the data dictionary.







There are to classifications of views :



•Because it is a simple view, INSERT, UPDATE, DELETE, and MERGE operations affecting the base table could possibly be performed through the view.

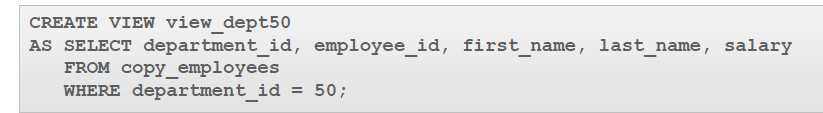
**Modifying a View**

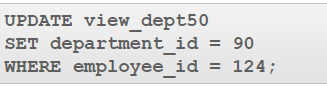
•To modify an existing view without having to drop then re-create it, use the OR REPLACE option in the CREATE VIEW statement.

•The old view is replaced by the new version.

**Views with CHECK Option**

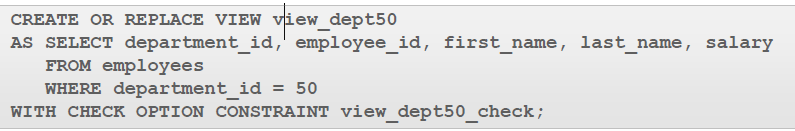
•The view is defined without the WITH CHECK OPTION.



•Using the view, employee\_id124 has his department changed to dept\_id90.

•The update succeeds, even though this employee is now not part of the view.

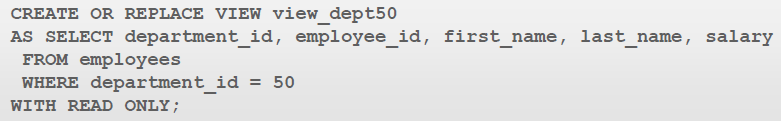
•The WITH CHECK OPTION ensures that DML operations performed on the view stay within the domain of the view.



**Views with READ ONLY**

•The WITH READ ONLY option ensures that no DML operations occur through the view.

•Any attempt to execute an INSERT, UPDATE, or DELETE statement will result in an Oracle server error.



**Deleting a View**

•Because a view contains no data of its own, removing it does not affect the data in the underlying tables.

•If the view was used to INSERT, UPDATE, or DELETE data in the past, those changes to the base tables remain.

•Deleting a view simply removes the view definition from the database.

•Only the creator or users with the DROP ANY VIEW privilege can remove a view.

