**SQL (Structure Query Language)**

**Data:** Data is a raw fact which describes the attributes of an entity.

**Data Base:** Data Base is a place or medium in which we store the data in a systematic and organized way.

Eg:- Facebook data base, oracle data base, drop box, google drive.

Basic operation done on a data base is

1. Create
2. Read/retrieve
3. Update
4. Delete

There are also known as CRUD operation.

**Data Base Management System (DBMS)**

Data Base Management system is a software which is used to manage the data base.

Security and authorization are the 2 most important features given by the Data Base Management System (DBMS)

Basic operation done in DBMS

1. Insert data
2. Read the data
3. Update the existing data
4. Delete the unwanted data

To communicate or interact with DBMS we use Query language.

Types of DBMS

1. Hierarchical
2. Relational
3. Network
4. Object oriented

**Relational Data Base Management System (RDBMS) :-**

Any DBMS which follows relational model is known as RDBMS.

To communicate with RDBMS we use structured Query Language (SQL)

**Relational Model:-**

Relation model was designed by “E.F CODD”.

In relational model we organize and store the data in the form of relations.

According to E.F CODD data in the relational model should be logically organized and stored in the form of tables.

**Tables: -** Table is logical organization of data. It consists of rows and columns.

– a table is a collection of rows and columns.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | **CELL** |  |
|  | **CELL** |  |

Rows /records /tuple

**Columns Attributes / field**

A table is also called as an entity / relation.

A cell is an intersection of a row and a column

**Column: -** Column is also known as attributes or fields. A column is used to represent an attribute of all the entities.

**Row: -** A row is also known as record or tuple.

A row is used to represent all the attributes of a single entity.

***Note* :-**

* If we install any of the database related software(s) – we can create our own database, we can create our own tables and we can store the data inside it.
* When we install any database s/w(s) – a part of hard disk will be designated / reserved to perform database related activities
* Some of the database software(s) we have are,

Oracle, SQL Server, DB2, Sybase, Informix, MySQL, MS – Access, Foxbase, FoxPro

Among the above database software – some of them are DBMS and some of them are RDBMS

The s/w which is widely used today is Oracle. The different versions of Oracle starting from the earliest to the latest are – Oracle 2, Oracle 3, Oracle 4, Oracle 5, Oracle 6, Oracle 7, Oracle 8i, Oracle 9i, Oracle 10g, and the latest to hit the market is Oracle 11g. here ‘i’ stands for Internet and ‘g’ stands for Grid / Grid computing.

**RELATIONSHIPS**

A relationship is the association between any two tables which preserves data integrity.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Relationships** | | | | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| **Master ( Parent )** | **Dept No.** | **Dept Name** |  |  | **Emp No.** | **Emp Name** | **Dept No.** | **Salary** | **Detail ( Child )** |
| 10 | Accounting |  |  | 101 | A | 10 | 20000 |
|  |  |  |  |  |  |  |  |
| 20 | Research |  |  | 102 | B | 10 | 300000 |
|  |  |  |  |  |  |  |  |
| 30 | Sales |  |  | 103 | C | 20 | 30000 |

Relationship helps to prevent the incorrect data in the child tables

Once the relationship is created, one table becomes master (or parent) and the other one becomes the child ( or detail ).

Whatever we insert into the child should be present in the master, else the record will be rejected from the child.

The master table contains the master data which will not change frequently.

The child table contains the transactional data which will change quite often

**DBMS**  & **RDBMS**

**DBMS** – stands for Database Management System

DBMS is a database s/w which allows us to store the data in the form of tables.

**RDBMS** – stands for Relational DBMS

RDBMS is also a database s/w which has facility to handle more data volume, good performance, enhanced security features etc when compared against DBMS.

Any DBMS to qualify as a RDBMS should support the Codd rules / Codd laws

**Ex** for DBMS – FoxPro, FoxBase, Dbase

**Ex** for RDBMS – Oracle, Sybase, DB2, Teradata, SQL Server, MySQL

**Data type:** -

It is an attribute that specify the type of data the object can hold.

**Char: -** char data type is used to store character, numerical and also special characters (A-Z, a-z, 0-9, !,@,#,$, .....)

Every time we define char data type we have to mention the size.

Fixed memory allocation type.

**Size: -** it is the maximum no. of characters that can be hold in char data type (maximum no. of character which it can hold is 2000)

**Syntax: char(size)**

EX: - char(10)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | P | I | D | E | R | S |  |  |  |

1 2 3 4 5 6 7 8 9 10

Used space memory wastage

**Varchar:-** Varchar data type is used to store characters numerical and also special characters (A-Z, a-z, 0-9, !,@,#,……)

Every time we define varchar data type we have to mention the size (maximum size is 4000)

Variable length memory allocation type.

**Syntax: - varchar(size)**

Ex: - varchar(10)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S | P | I | D | E | R | S |  |  |  |

1 2 3 4 5 6 7 8 9 10

Used space free memory

**Number :-** Number data type is used to store only numerical data type.

It can accept only 2 arguments

**Syntax: number(precision, scale)**

Ex: - number(4)

Precision specifies only the no. of digits needed and the range is -9999 to 9999, maximum limit of precision 38.

EX: - number(4, 1)

Scale specifies the no. of digits it needs to store the decimal value. Maximum limit for scale 127 -999.9 to 999.9

Ex:- number(6, 2)

-9999.99 to 9999.99

**Date: -** Date data type is a format given by oracle

DD/MM/YYYY or MM/DD/YYYY

**Large object:-**

1. **Character large object (CLOB) :-** it is use to store character up to 4GB of size.
2. **Binary large object (BLOB):-** Binary Large Objects are used to store images, MP3’s, MP4’s, documents etc., up to 4 GB of size.

**CONSTRAINTS**

A constraint are the rules which to be satisfy before the data is entered into the table

A constraint is a condition which restricts the invalid data in the table.

A constraint can be provided for a column of a table.

**Types of Constraints**

* NOT NULL
* UNIQUE
* Primary Key
* Foreign Key
* Check

**Characteristics of NULL**

* NULL is nothing, Null is not equal to zero or space
* It will not occupy any space in the memory
* Two NULLS are never same in Oracle.
* NULL represents unknown value
* Any arithmetic operation we perform on NULL will result in NULL itself. **For ex,** 100000 + NULL = NULL ; 100000 \* NULL = NULL

**NOT NULL**

- NOT NULL will ensure at least some value should be present in a column

**UNIQUE**

* It will not allow any duplicates in a column
* UNIQUE column can take multiple NULL (s)

**Primary Key**

* It is the combination of **NOT NULL** and **UNIQUE**
* Only one PK is allowed in a table
* PK identifies a record uniquely in a table
* Creation of PK is not mandatory, but it is highly recommended to create

**Foreign Key**

* FK creates relationship between any two tables
* FK is also called as referential integrity constraints
* FK is created on the child table
* FK can take both NULL and duplicate values
* To create FK, the master table should have PK defined on the common column of the master table
* We can have more than 1 FK in a given table

**CHECK**

It is used to provide additional validations as per the customer requirements.

**Ex -**  1) sal > 0

2) empnum should start with 1

3) commission should be between 1000 & 5000

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  | **Check (sal > 0)** |  |
|  |  |  |  |  |  |  |  |  |  |
| **PK** | **NN** |  |  | **PK** | **NN** | **FK** | **NN** | **Unique** | **Unique** |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| **Dept No.** | **Dept name** |  |  | **Emp No.** | **Emp Name** | **Dept No.** | **Salary** | **Ph No.** | **Email** |
| 10 | Accounting |  |  | 101 | A | 10 | 200000 | 2222 | [a@gmail](mailto:a@gmail) |
|  |  |  |  |  |  |  |  |  |  |
| 20 | Research |  |  | 102 | B | 10 | 30000 | - | - |
|  |  |  |  |  |  |  |  |  |  |
| 30 | Sales |  |  | 103 | C | 20 | 400000 | 3333 | - |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| **RELATIONSHIP** | | | | | | |  | **NULL** | |

**History of SQL**

In the early 80’s relational model was quite popular, IBM used relational model and develop an RDBMS.

To communicate or to interact with system they developed Query language called SEQUEL (simple English query language)

All the industries started using SEQUEL, then ANSI (American National Standard Institute) took SEQUEL they made it a standard language.

SEQUAL was renamed as structured query language (SQL)

SQL was made the standard language to communicate with RDBMS.

SQL – it is a language to talk to the database / to access the database

SQL – it is a language, whereas SQL server is a database.

To work on SQL , a DB software (RDBMS) is required.

SQL is not case sensitive

**Username**  - Scott

**Password** – Tiger

**Projection: -**

SQL statement consist of multiple clause, each clause is a sub program which can accept argument as input.

**Syntax: select \* /[distinct] column name/ expression[alias]**

**From <table\_name>;**

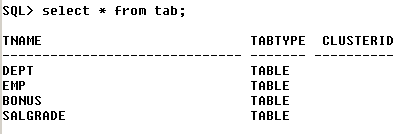
1. From clause will execute first.
2. For from clause we can pass table name as an argument.
3. Select clause will execute after from clause.
4. For select clause we can pass asterisk symbol, column name or expression as an argument.
5. Select clause is used to select a column or expression from the table which is achieved by from clause.
6. Select clause is responsible to prepare the result set

Expression:

Expression is something which gives result.

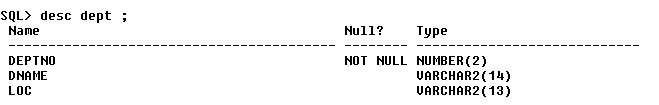
**Selection: -**

* Selection is a processer retrieval of data by selecting both rows as well as columns.
* From clause is use to select the table from the data base and put it under execution
* Select clause is use to select the column present in the table which is under execution.
* There is a clause which is use to select the records that is where clause.

****

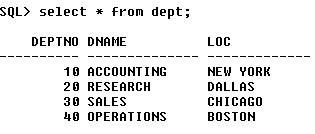
This query gives the list of tables.

\* - selects all

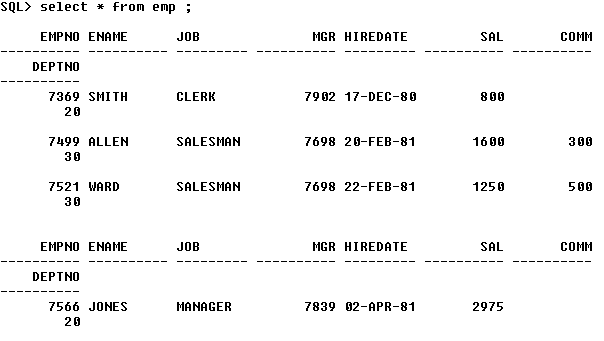


This query gives the description of the table “department”.

The description of the table has **column names, constraints, datatypes**

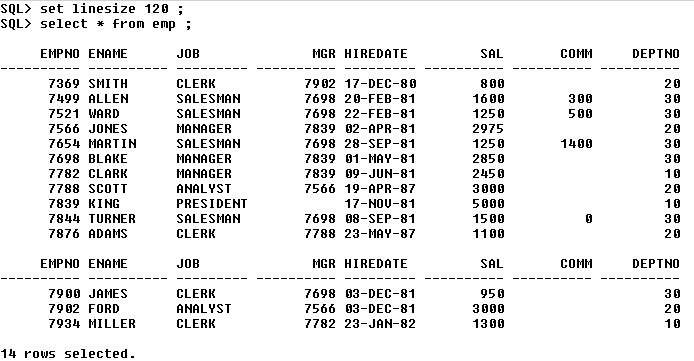


This query gives the description of the table “department”

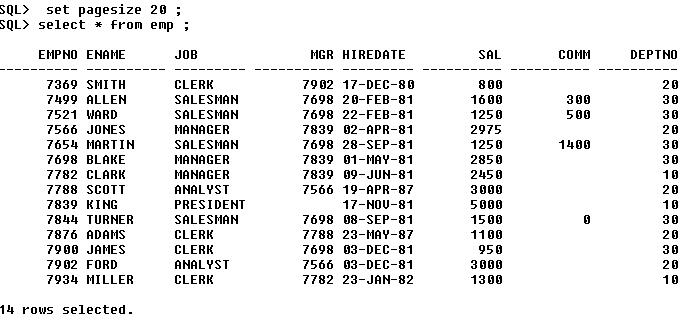


The above query gives the description of the “employee” table. But we see that all the data is in different lines which makes it very difficult to analyse.

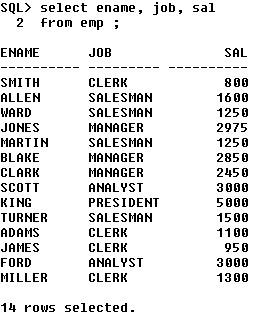
So we use the following command to see the data in a more orderly fashion,



The **“set linesize”** command helps in increasing the line size , thus the data is arranged in a orderly fashion.



The above command **“set pagesize 20”** increases the page size, thus accommodating more number of rows in a single page.



The above query gives the value of only these 3 columns from the table “employee”.

**Where clause: -**

**‘where’** clause is used to restrict the number of records displayed. It gives only the records of the specified condition.

or

Where clause is used to filter the records present in the table.

**Syntax: select \* /[distinct] column /expression [alias]**

**From < table\_name>**

**Where <filter condition>;**

**Order of execution**

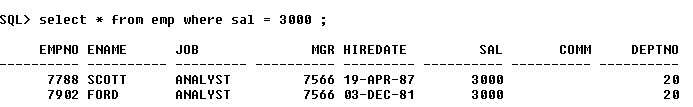
1. **From clause**
2. **Where cause**
3. **Select clause**

* **Where clause execute row by row**
* **Where clause can accept multiple conditions**
* **For where clause we can pass filter condition or multiple condition**

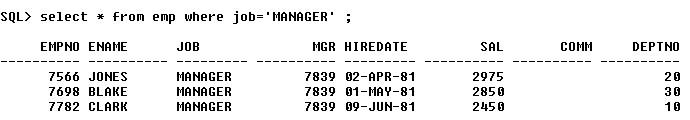
Ex: - select \*

From emp

Where sal>200;

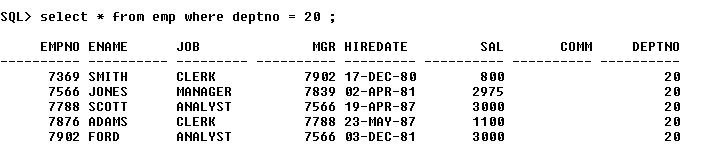


Any string data should be enclosed within **single quotes** ( ‘ ‘ ) and the same becomes **case sensitive**.

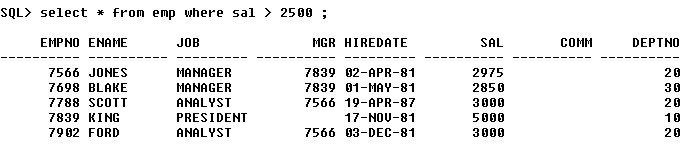


**Assignment**

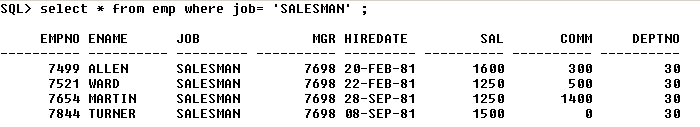
**1) List the employees in dept 20**

****

**2) List the employees earning more than Rs 2500.**



**3) Display all salesmen**



**Questions**

1. WAQTD name, salary and annual salary of an employee who is working as sales man.
2. WAQTD ename, job, hired date of all employes who is working in department no. 10
3. WAQTD ename, job, MGR no. of all the employee who are working in department 30 and having salary of 3000
4. WAQTD ename, dept no. comm. And sal of smith.
5. WAQTD name of an employee who is working in department 10 and having salary greater than 1500
6. WAQTD name of an employee who is having salary greater than 950 and working as cleark
7. WAQTD department name and location from department table
8. List the employees in dept 20
9. List the employees earning more than Rs 2500.
10. Display all salesmen

**Operators**

Operators are classified into,

* **Arithmetic Operators** ( +, - , \* , / )
* **Relational Operators** ( > , < , >= , <= , = , <> or != - not equals to )
* **Logical Operators** ( NOT, AND, OR )
* **Special Operators** ( IN , LIKE , BETWEEN , IS )

**NOT IN, NOT LIKE, NOT BETWEEN, NOT IS**

**Questions**

1. WAQTD name, salary and job of all the employees working as clerk with a salary greater than 1500.

Ename, Salary, JOB

JOB = clerk

Sal>1500

**Select ename, sal, job**

From emp

Where sal>1500 AND JOB=’CLERK’;

1. WAQTD name, hire date and salary of the employees who gets a salary greater that 1250 and hired before 19-nov-82
2. WAQTD name and designation of all those employees who were hired after 1981 and before 1986

Select ename, job,

From emp

Where hireddate>31 dec -81 and hiredate< 01-jan-86

1. WAQTD all the details of employees who work as sales man or manager in department no 10 or 20

(Sales man or manager) and ()

Dept no = 10 or 20

Sales man or manager in dept 10 or 20

Select \*

From emp

Where (job=’salesman’ or job= ‘ manager’)and (deptno.= 10 or deptno.= 20);

1. Display “Mr ’ENAME’ gets a salary of rupees ‘SAL’ and works as ‘JOB’ in department no. ‘DEPTNO’”

* **Here we use concatenation operator (||)**

Select “’Mr ‘||ename||’gets a salary of rupees ‘||sal||’ and works as’||job||’in dept no’||deptno.|| “

From emp;

1. WAQTD details of all employees who work as clerk or manager or salesman in department no. 10 or 20 or 30

Select \*

From emp

Where job IN (clerk, manager,salesman) and dept no. IN (10,20,30);

1. WAQTD ename, dept. no. and designation of all the employees who are reporting to 7782

**Special Operators**

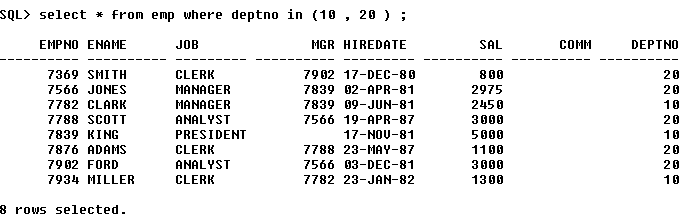
**IN operators:**  it is used for evaluating multiple values.

In operator is a multi value operator which can accept one value at the LHS and multiple values at RHS.

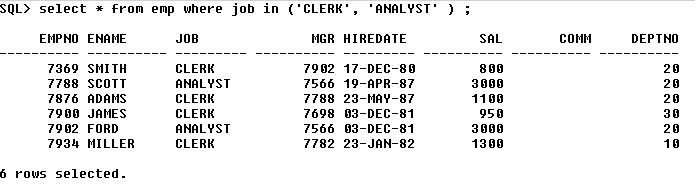
We can provide upto 1000 values at the max

**Syntax: column name IN (v1, v2, v3, …….,vn);**

**Ex – 1)** List the employees in dept 10 or 20



**2)** List all the clerks or analysts



1. WAQTD name and job of a employee who work in dept. 10 or 20 or 40

**2) LIKE** – used for pattern matching

**% (percentage) -** matches 0 or ‘n’ characters

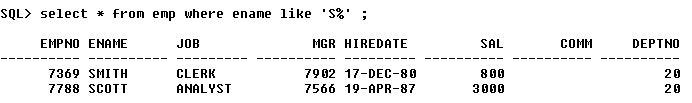
**\_ (underscore)** - matches exactly one character

**Ex – 1) List all the employees whose name starts with ‘S’**

Select \*

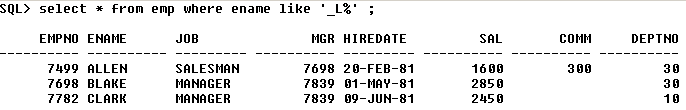
From emp

Where ename like ‘s%’



Whenever we use % or \_, always ensure that it is preceded by the word **‘like’**

**2) List the employees whose name is having letter ‘L’ as 2nd character**



ASSIGNMENT

1. List the employees whose name is having at least 2 L’s
2. List the employees whose name is having letter ‘E’ as the last but one character
3. List all the employees whose name is having letter ‘R’ in the 3rd position
4. List all the employees who are having exactly 5 characters in their job
5. List the employees whose name is having at least 5 characters

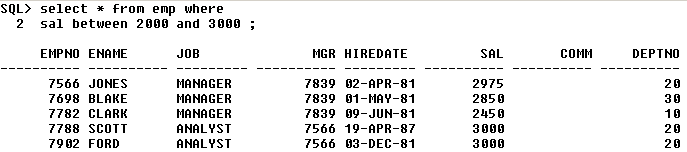
**3) BETWEEN** operator – used for searching based on range of values.

Select \*

From emp

Where sal between 200 and 300;

**Ex – 1)** List the employees whose salary is between 200 and 300



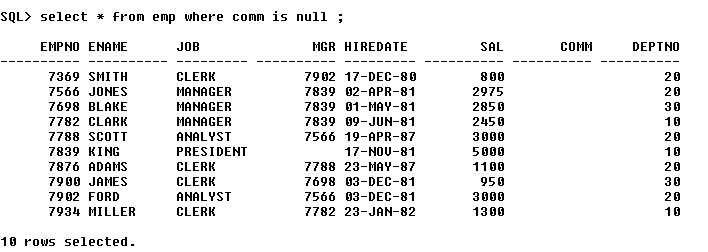
**4) IS operator – it is used to compare null values**

**Ex – 1)** List all the employees whose commission is null

Select \*

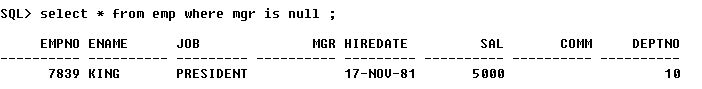
From emp

Where comm IS null;



**ASSIGNMENT**

**1) List all the employees who don’t have a reporting manager**



**LOGICAL OPERATORS**

1) List all the salesmen in dept 30

2) List all the salesmen in dept number 30 and having salary greater than 1500

3) List all the employees whose name starts with ‘s’ or ‘a’

4) List all the employees except those who are working in dept 10 or 20.

5) List the employees whose name does not start with ‘S’

**Select \***

**From emp**

**Where ename not like ‘s%’**

**6) List all the employees who are having reporting managers in dept 10**

**Select \***

**From emp**

**Where mgr not IS null and dept no.=10;**

**ASSIGNMENT**

1. List the employees who are not working as managers and clerks in dept 10 and 20 with a salary in the range of 1000 to 3000

JOB = Manager and clerks

Dept no = 10 and 20

Sal 1000 to 3000

Select \*

From emp

Where (job = ‘manager, job=’clerk’) and (deptno=10, deptno=20) and sal between 1000 to 3000;

1. List the employees whose salary not in the range of 1000 to 2000 in dept 10,20,30 except all salesmen

Sal not in between 1000 to 2000

Dept no = 10, 20,30

Job <> saleseman

Where (sal not between 1000 and 2000) and (deptno in (10,20,30)) and (job <> salesman));

1. List the department names which are having letter ‘O’ in their locations

Select \*

From dept

Where loc like ‘%o%’;

**ALIAS: -**

* Alias is a name given to a expression or to the column present in the result table.
* We can write alias name with or without using **as** key work
* If column name requires a space between two words then it is mandatory to enclose the name with double cote (“ ”)
* Syntax:

Select <\*/column/expression> as <alilas name>

From <table name>;

Q. WAQTD name of an employ, his salary as Monthly salary and his designation of all the employees.

Select Ename, Sal as “SALARY of an Employee”, job

From emp;

Q. WAQTD name of an employ, his annual salary as Annual Salary and his designation for all the employees.

Select Ename, sal\* 12 as “Annual Salary”, job

From emp;

**SORTING**

It arranges the data either in ascending / descending order

Ascending – ASC / Descending – DESC

We can sort the data using **ORDER BY**

By default, the data is always arranged in ASC order

**Syntax:**

Select <\*/column/expression>

From <table name>

Where <filter condition>

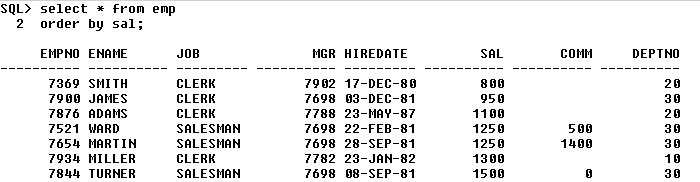
Order by <column name> asc/desc;

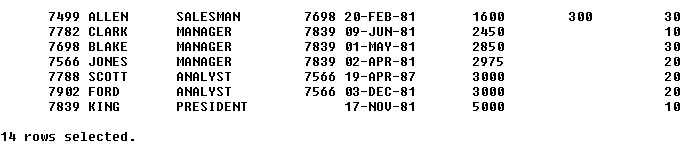
**For ex – 1) Arrange all the employees by their salary**

**Select \***

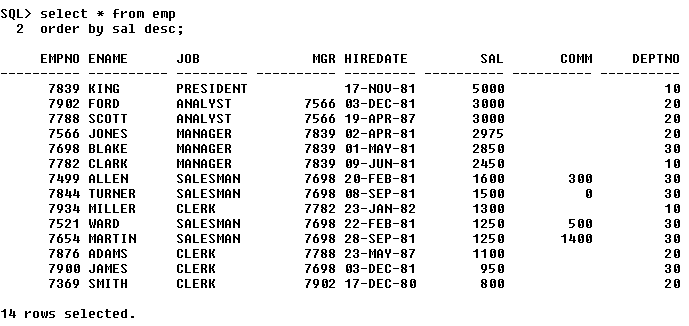
**From emp**

**Order by sal asc;**



****

**2) Arrange all the employees by their salary in the descending order**

****

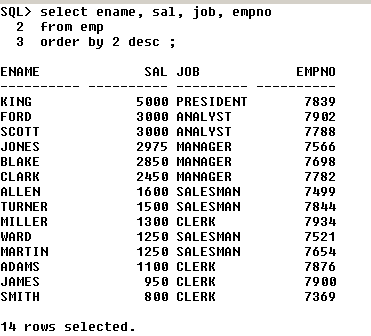
**3) Arrange ename, sal, job, empno and sort by descending order of salary**

**Select ename, sal, job, empno**

**From emp**

**Order by 2 desc;**

1. **From**
2. **Select**
3. **Order by**

****

In the above query we have – **order by 2** – thus it arranges only the 2nd column ‘salary’ in the descending order.

Thus to arrange the specific columns in order – we must have to specify the column number.

***NOTE :-*  ORDER BY** should be used always as the last statement in the SQL query.

**Distinct clause:**

Distinct clause is used to remove the duplicate records from the column.

**Syntax :**

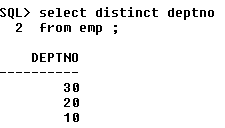
Select distinct <column name>

From <table name>;

Select distinct sal

From emp

**Selecting DISTINCT VALUES**



The above query arranges all the distinct values of department number.

**Functions**

**Function is a block of code which is used to perform particular operation or task**

**Any functions involves 3 things**

1. Function name
2. No. of argument / type of argument
3. Return type

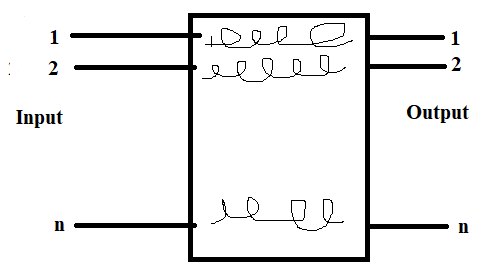
There are 2 types of functions in SQL

1. Single row function
2. Multi row function or group function or aggregate function

**Single row function:**

Single row function is the functions which will execute for each and every row and generate results or output for each row.

If n No. of input’s are given the function will generate ‘n’ No. of output

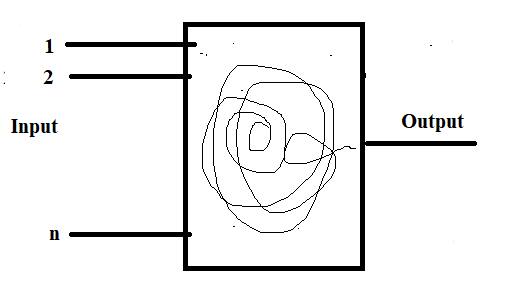


In single row function executes row by row.

**Multi row function:**

Multi row function is the function which will aggregate (combine) all the inputs and executes only one hence aggregating one output.

If there are n No. of inputs multi function will written single output



**List of multi row functions**

1. MAX() – returns maximum value
2. MIN() –returns minimum value
3. SUM() – returns total value
4. AVG() – returns average value
5. COUNT() – returns no. of records

Count is a multi row function for which we can pass any column name.

**Note:** Multi row functions will ignore null values

**Ex –**

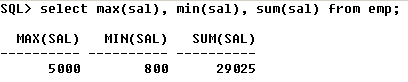
1. **display the maximum salary, minimum salary and total salary of all the employee**

**select max(sal) as “Maximum Salary”, min(sal) as “Minimum Salary “, sum(sal) Total**

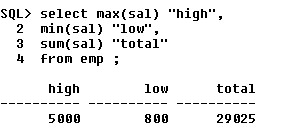
**from emp;**

**select count (empno) as “Total No. of employees”**

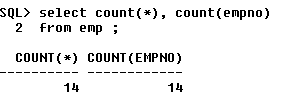
**form emp**



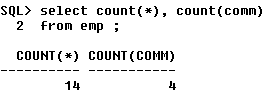
***To give aliases for the columns*** *:-*



**3) The below query gives the total number of employees**



**4) The below query gives the number of employees who have commission**

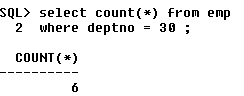
****

1. **List the number of employees in department 30**

**Select count(\*)**

**From emp**

**Where deptno=30;**

****

**Assignment:**

1. WAQTD maximum salary of an employee and his name, job who is working as salesman
2. Display the total salary in department 3
3. List the number of clerks in department 20
4. List the highest and lowest salary earned by salesmen
5. WAQTD minimum salary of an employee who works in Dept. No. 20 or 30.
6. WAQTD average salary needed to pay all the employees who are working as a clerk.
7. WAQT find the total salary of all the employees who are working as analyst or president.
8. WADTD No. of people who are working as salesman with a salary of more than 1500.

**GROUPING**

Group by clause:

1. Group by clause is used to group the records present in the table
2. Group by clause executes row by row
3. After the execution of group by clause, the records are grouped.
4. After the execution of group by clause all the other clause will execute group by group.
5. In select clause we can use only group by expression and multi row functions which will be executed after group by clause.

Syntax: select <group\_by expression / multi row function>

From <table name>

Where <filter condition>

Group by <column name or expression>;

Example:

Select class, count(\*)

From student

Group by class;

11

|  |
| --- |
| 1 A 11  6 F 11 |

|  |  |
| --- | --- |
| class | Count() |
| 11 | 2 |
| 12 | 2 |
| 10 | 2 |

|  |
| --- |
| 2 B 12  3 C 12 |

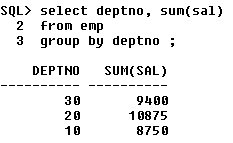
12

|  |  |  |
| --- | --- | --- |
| SID | Sname | class |
| 1 | A | 11 |
| 2 | B | 12 |
| 3 | C | 12 |
| 4 | D | 10 |
| 5 | E | 10 |
| 6 | F | 11 |

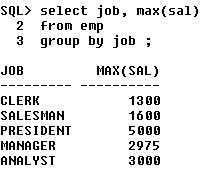
|  |
| --- |
| 4 D 10  5 E 10 |

10

**For ex – 1) Display the total salary of all departments**

****

**2) Display the maximum salary of each job**

****

**HAVING**

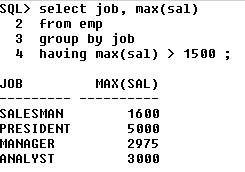
**‘Having’** is used to filter the grouped data.

**‘Where’** is used to filter the non grouped data.

‘**Having’** should be used after **group by** clause

**‘Where’** should be used before **group by** clause

**For ex – 1) Display job-wise highest salary only if the highest salary is more than Rs1500**

****

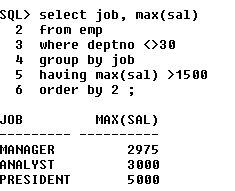
1. **Display job-wise highest salary only if the highest salary is more than 1500 excluding department 30. Sort the data based on highest salary in the ascending order.**

**Group by job**

**Max(sal)>1500**

**Deptno<>30**

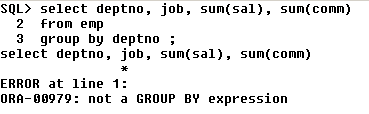
**Orderby sal**

****

**RESTRICTIONS ON GROUPING**

- we can select only the columns that are part of ‘**group by’** statement

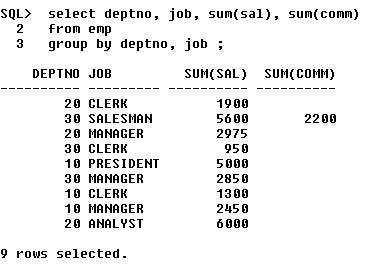
If we try selecting other columns, we will get an error as shown below,

****

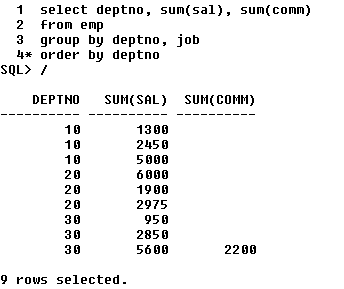
The above query is an error because ‘**job**’ is there in the **select** query but not in the **group by** query.

If it is enclosed in any of the **group functions like sum(sal)** etc – then it is not an error. But whatever table is included in the **select** query must also be included in the **group by** query.

The above problem can be overcome with the following query as shown below,



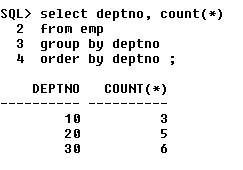
The below query is also correct to rectify the above error,



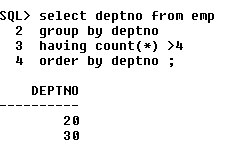
Whatever is there in the **select** statement must be there in the **group by** statement. But, whatever is there in the **group by** statement need not be present in the **select** statement. This is shown in the above two corrected queries.

**ASSIGNMENT**

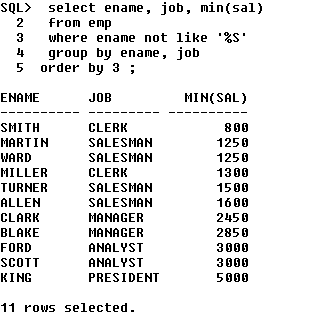
**1) Display the department numbers along with the number of employees in it**



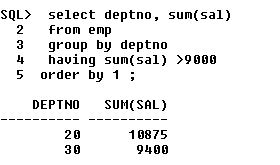
**2) Display the department numbers which are having more than 4 employees in them**

****

**3) Display the maximum salary for each of the job excluding all the employees whose name ends with ‘S’**

****

**4) Display the department numbers which are having more than 9000 as their departmental total salary**

****

**Difference between where and having clause**

|  |  |
| --- | --- |
| **Where** | **having** |
| 1. Where clause is used to filter the records in the table | 1. Having clause is used to filter the groups |
| 1. Where clause execute row by row | 1. Having clause execute group by group |
| 1. We cannot use multi row function in where clause | 1. We can use multi row function |
| 1. Where clause execute before group by | 1. Having clause execute after group by |
| 1. In where clause we can write any filter condition | 1. In having we can have only group by expression and conditions with multi row function |

Order of execution

1. From
2. Where
3. Group by
4. Having
5. Select

**Sub Query:**

A query written inside another query is known as sub query.

Working principle / Procedure

Result

Dependent

Input

Output

1. The inner query will execute first
2. The inner query generates an output, which is given as an input to the outer query.
3. With this input the outer query executes completely. Hence, generates the result.
4. The outer query cannot execute without the inner query (the outer query is dependent on inner query)

When do we use sub query?

* Case I :- whenever we have unknown we go for sub query

Example : - WAQTD details of all the employees whose salary is greater that FORD salary

Details of all the employees

Sal > fords sal

Select \*

From emp

Where sal > (select sal

From emp

Where ename=’FORD’ );

* Case ii : - whenever the condition to be executed is in one table and the data to be displayed is from another table

Example :- WAQTD Dept. name of SMITH

Select dname

From dept

Where deptno= (select deptno

From emp

Where ename=’SMITH’);

Questions: -

1. WAQTD name and dept no. of all the employees who are working in the same dept. in which JONES is working

* Ename, Dept NO.

Select Ename , deptno

From emp

Where deptno = (Select deptno

From emp

Where ename =jones);

1. WAQTD name and job of all the employees who are working in the same designation in which SMITH works.
2. WAQTD name, hire date of all the employees who were hired after and before WARD
3. WAQTD dept.name of all the employee whose salary is greater than 1300.
4. WAQTD names of all the employees who are working in sales dept.
5. WAQTD details of employees whose dept name ending with ‘s’
6. WAQTD the employees name who is having maximum salary in dept. name ‘ACCOUNTING’

**Types of sub query**

There are 2 types of sub query

1. Single row sub query
2. Multi row sub query

**Single row sub query: -**

If the inner query written exactly one record then it is called as single row sub query.

In single row sub query all the comparison operators can be used (=, >, <, >=, <=, <>)

**Multi row sub query: -**

If the inner query written more than one record then it is called as multi row sub query.

In multi row sub query we cannot use the comparison operator directly therefore we have 2 operators they are

1. ALL
2. ANY

ALL : -

ALL operators is a multi valued relational operator which will written true only if all of the values that is compared is true.

If any of the value is false then it written false

Syntax: **column\_name relational\_operator ALL(v1, v2, v3, …….vn)**

ANY: -

ANY operator is a multi valued relational operator it will write true if any of the values that has compared is true.

Syntax: **column\_name relational\_operator ANY(v1, v2, v3, …….vn)**

**Nested Sub Query**

We can nest a sub query to the where clause 255 times

Q. WAQTD 4th maximum salary of the employees

Select max(sal)

From emp

Where sal< (Select max(sal)

from emp

Where sal< (Select max(sal)

From emp

Where sal< (Select max(sal)

From emp) ));

(9, 5, 4, 1, 7, 3, 2, 6, 8)

Select max(sal)

From emp

Where sal<(10

questions.

1. WAQTD 2nd min salary of an employee
2. WAQTD 5th max SALARY of an employee
3. WAQTD hire date of all the employees working in location ‘DALLAS’
4. WAQTD all the details of employee who are reporting to ‘KING’
5. WAQTD details of all the employees who are acting as manager
6. WAQTD name of all the employees who gets 3rd max sales man among all the sales mans

**JOIN**

Merging of two or more tables horizontally is known as Joins

Q. Why do we need Join’s?

🡺 To retrieve the data from multiple tables we use join’s.

When we have to retrieve the data from 2 tables then we perform join.

Note: - from clause is responsible to merge the table

Types of Join’s

1. Cartesian join or cross join
2. Inner join or equi join
3. Outer join
4. left outer join or left join
5. right outer join or right join
6. full outer join or full join
7. self join
8. **Cartesian join or cross joins: -**

If we join 2 tables, records from one table is merged with each and every records present in the other table is known as Cartesian join or cross join.

Ex. Let us consider 2 tables T1 and T2 with columns each and m, n as no. of rows respectively

T1 T2

|  |  |
| --- | --- |
| **A1** | **B1** |
| A | 10 |
| B | 20 |
| C | 30 |

|  |  |
| --- | --- |
| **A2** | **B2** |
| B | 200 |
| C | 300 |
| D | 400 |

If we perform a Cartesian join on table T1 and T2 the newly obtain table will have 4 columns and m X n no. of rows.

Note: -

1. Cartesian join has valid as well as invalid pairs.
2. Cartesian join as a universal set as it is having all the possible combinations.

T1 X T2 only when A2 = B

|  |  |  |  |
| --- | --- | --- | --- |
| A1 | B1 | A2 | B2 |
| A | 10 | B | 200 |
| A | 10 | C | 300 |
| A | 10 | D | 400 |
| B | 20 | B | 200 |
| B | 20 | C | 300 |
| B | 20 | D | 400 |
| C | 30 | B | 200 |
| C | 30 | C | 300 |
| C | 30 | D | 400 |

m X n = 3 X 3 = 9

Syntax for Cartesian join: -

1. ANSI syntax : -

Select \*/column/expression

From table1 cross join table2;

Eg. Select \*

From T1 cross join T2;

1. Oracle syntax:-

Select \*/column/expression

From table1, table2,…..;

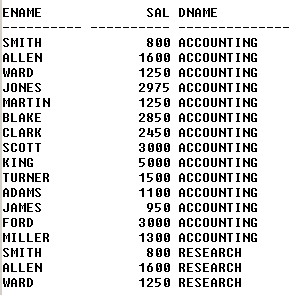
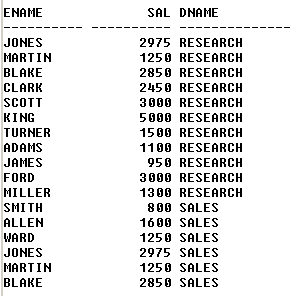
Eg. Select \*

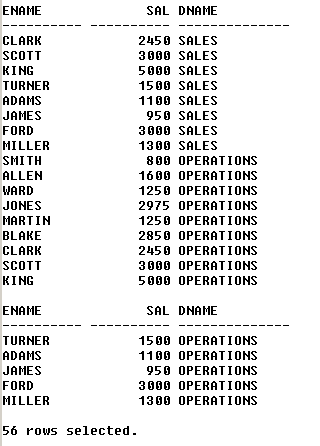
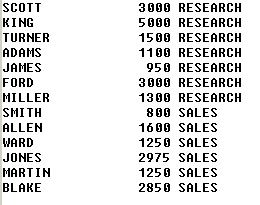
From T1, T2, T3,….;

**For ex,** let us consider the following query

**Display employee name along with the department name**





From above – we can see that the above query returns 56 records – but we are expecting 14 records. This is because each and every record of employee table will be combined with each & every record of department table.

Thus, Cartesian join should not be used in real time scenarios.

The Cartesian join contains both correct and incorrect sets of data. We have to retain the correct ones & eliminate the incorrect ones by using the **inner join**.

1. **Inner join: -**

Inner join are also called as **equijoins**.

They return the matching records between the tables.

In the real time scenarios, this is the most frequently used Join.

We join two tables such that a record from one table is merged to a record from another table only when given condition is satisfied is known as inner join.

**For ex,** consider the query shown below,

Select A.ename, A.sal, B.dname

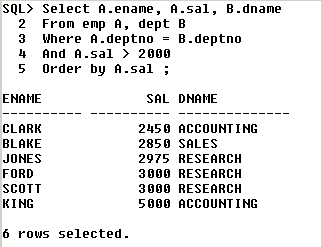
From emp A, dept B

Where A.deptno = B.deptno - **JOIN condition**

And A.sal > 2000 **- FILTER condition**

Order by A.sal ;

Let us see the output shown below,



JOIN condition is mandatory for removing the Cartesian output.

Let us consider the following 2 scenarios shown below,

**Scenario 1**

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | | |  |
| **P** | **Q** | **R** |  |
|  |  |  |  |
|  |  |  |  |
| **B** | | |  |
| **P** | **S** | **T** |  |
|  |  |  |  |
|  |  |  |  |
| **C** | | |  |
| **P** | **X** | **Y** |  |
|  |  |  |  |
|  |  |  |  |
| **We want** | | | |
| **P** | **Q** | **S** | **X** |
|  |  |  |  |

**The SQL query will be,**

**Select** A.P, A.Q, B.S, C.X

From A, B, C

Where A.P = B.P **Number of joins = 2**

And A.P = C.P

**Therefore, Number of JOINS = Number of tables - 1**

**Scenario 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | | |  |  |
| **P** | **Q** | **R** |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **B** | | | |  |
| **P** | **Q** | **S** | **T** |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **C** | | |  |  |
| **P** | **X** | **Y** |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **We want** | | | | |
| **P** | **Q** | **R** | **S** | **X** |
|  |  |  |  |  |

The **SQL query is ,**

Select A.P, A.Q, A.R, B.S, C.X

From A, B, C

Where A.P = B.P

And A.Q = B.Q **Number of Joins = 3**

And A.P = C.P ;

**Therefore, Number of JOINS = Number of common columns**

If there are no common columns, then reject it saying that the two tables can’t be joined.

But there are some cases – where the 2 columns will be same but having different column names.

**For ex –** customerid & cid

ANSI Syntax:

Select \*

From table1 inner join table 2

ON <join condition>

Where <filter condition>

Thus we, can see the changes ,

* In the 2nd line - ,(comma) has been replaced by the word ‘join’
* In the 3rd line – ‘where’ has been replaced with ‘on’

Note :

1. To perform inner join, join condition is mandatory
2. Join condition: - it is a condition which includes column from both the tables
3. Inner join is a sub set of Cartesian join or cross join

Ex. Let us consider the table T1 And T2 we join T1 and T2 using the join condition

T1.A1 = T2.A2

The table obtained is as follows

|  |  |  |  |
| --- | --- | --- | --- |
| A1 | B1 | A2 | B2 |
| B | 20 | B | 200 |
| C | 30 | C | 300 |

Q. WAQTD Dept name, salary, comm of all the employees who are working in accounts or research dept. as a manager

Select dname, salary, comm.

From emp, dept

ON emp.deptno=dept.deptno

Where job=’manaager’ and (dname = accounts or dname=’’research)

ANSI

Select dname, sal, comm

From emp inner join dept

ON emp.deptno = dept.deptno

Where dname IN (‘account’,’research’) and job =’manager’;

Select dname, sal, comm.

From emp, dept

Where emp.deptno=dept.deptno

And dname in (‘account’,’research’) and job =’manager’;

Q. WAQTD dept name, ename, sal of all the employee whose name starts with A whose dept name ends with S and having the salary between 3000 and 5000

Dept Name, Ename, SAL

Condition

Ename starts with A and

Dname ends with S and

sal between 3000 and 5000

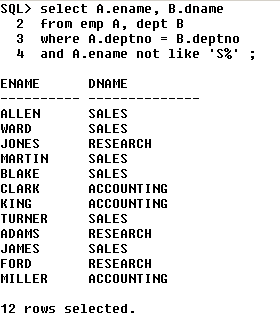
select detname, ename, sal

from emp, dept

where emp.deptno=dept.deptno and

**Assignment**

**1) Display employee name and his department name for the employees whose name starts with ‘S’**

****

**Outer Join: -**

It returns both matching and non-matching records

Outer join = inner join + non-matching records

Non-matching records means data present in one table, but absent in another table w.r.to common columns.

**For ex,** 40 is there in deptno of dept table, but not there in deptno of emp table.

Dept

Emp

|  |  |
| --- | --- |
| Ename | Dept no. |
| 1 | 10 |
| 2 | 20 |
| 3 | 10 |
| 4 | 10 |
| 5 | 40 |

|  |  |  |  |
| --- | --- | --- | --- |
| Dname | | Dept No. | |
| D1 | | 10 | |
| D2 | | 20 | |
| D3 | | 30 | |
| Dname | Dept NO. | ename | | Dept no. | |
| D1 | 10 | 1 | | 10 | |
| D1 | 10 | 3 | | 10 | |
| D1 | 10 | 4 | | 10 | |
| D2 | 20 | 2 | | 20 | |
| D3 | 30 |  | |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| Ename | Dept NO. | Dname | Dept no. |
| 1 | 10 | D1 | 10 |
| 2 | 20 | D2 | 20 |
| 3 | 10 | D1 | 10 |
| 4 | 10 | D1 | 10 |
| 5 | 40 |  |  |

**Left outer join:** - left outer join is used to obtain the unmatched of left table

Select \*

From table1 left outer join table 2

On <join condition>

Where <filter conditioin>;

Note:

1. To get only unmatched records from the left table we should write a condition that is

R\_table\_name.column\_name IS null;

1. To get only unmatched records from right table we should write the condition that is

L\_table\_name.columnname IS null;

Q. WAQTD name of an employee who is not working in any department

Select ename

From emp left outer join dept

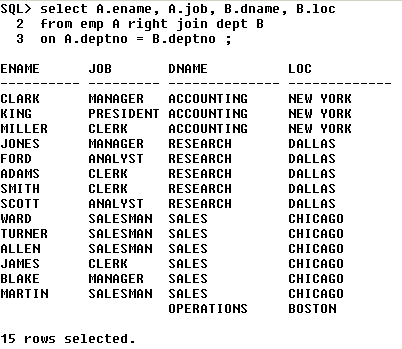
ON emp.deptno=dept.deptno

Where dept.deptno = null

2. Right outer join: - it is used to obtain the unmatched records go the right table

**Display all the department names irrespective of any employee working in it or not. If an employee is working – display his name.**

***Using right join***

****

***Using left join***

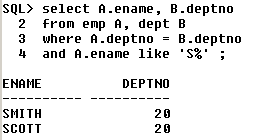
****

***Using full join***

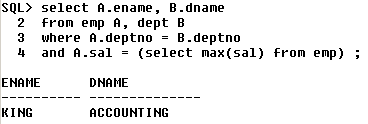
****

**Assignment**

**1) Display employee name and his department name for the employees whose name starts with ‘S’**

****

**2) Display employee name and his department name who is earning 1st maximum salary**

****

**SELF JOIN**

Self join used to obtain the data to be selected in the same record or row

Joining a table to itself is called self join

The **FROM** clause looks like this,

FROM emp A, emp B

Or

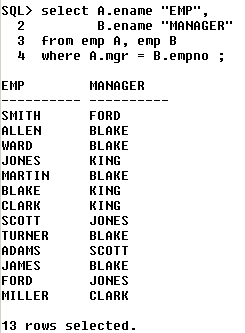
FROM emp A join emp B - *ANSI style*

For ex, - **Display employee name along with their manager name**

**Select a.ename, b.ename**

**From emp A, emp B**

**Where A.mgr=b.empno**

****

**Now, let us see how this i.e the logic (the above query) works,**

|  |  |  |
| --- | --- | --- |
| **Emp (A)** | | |
| **EmpNo** | **Ename** | **Mgr** |
| 101 | Scott | 102 |
| 102 | Blake | 103 |
| 103 | King | - |
| 104 | Smith | 103 |
| 105 | Jones | 104 |

|  |  |  |
| --- | --- | --- |
| **Emp (B)** | | |
| **EmpNo** | **Ename** | **Mgr** |
| 101 | Scott | 102 |
| 102 | Blake | 103 |
| 103 | King | - |
| 104 | Smith | 103 |
| 105 | Jones | 104 |

Now, when we give the above query – in Oracle – it starts matching the ‘**mgr**’ column of **emp A** with the ‘**empno**’ of **emp b** – we get two tables because in **self join** – a duplicate of the table required is created.

Now let us consider the **first employee Scott** – it starts the **mgrid** of **Scott** with the **empno** of all the records in **emp B** – when two **ids** match, then the **empno** in **emp B** becomes the **mgr** of the **empno** in **emp A**. Thus, we can see that – **mgr id** 102 is matching with **empno** 102 **Blake** in **emp B**. Therefore, Blake is the manager of Scott.

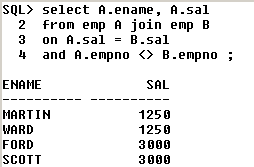
Similarly we do the same for all the other records of **emp A** and thus find the employees and their respective managers.

**Display the employees who are getting the same salary**

Select a.ename, a.sal

From emp a, emp b

Where a.sal=b.sal



**Co –related Sub-query**

In Co-related sub queries the outer query is dependent on inner query and inner query is also dependent on outer query, this is called Co-related sub query

In Co-related sub queries the inner should consists the condition which includes a column from the outer query.

This introduces the dependence between the inner query and the outer query.

Q WAQTD a Dname where at least one employee is working

Select dname

From dept

Where deptno IN (select deptno

From emp

Where dept.deptno=emp.deptno);

Co –related sub query were under the principle of sub-query and join

The execution flow of co-related subqueries

1. Outer query gets partially executed
2. If a record in the outer query is selected or rejected it is purely depends on the output generated by the inner query.
3. Since the inner query depends on outer query, inner query gets completely executed for each and every records present in the outer query.

Q . WAQTD a Dname where there are no employees are working

Q WAQTD dname where at least 2 employees are working

**Using the sub query in from clause**

We can use sub query in from clause.

The output of the sub query (result set table) is considered as the table for execution.

The processes of using a sub query in from clause are also known as Inline Views.

Ex.

Select \*

From (select \*

From emp

Where deptno=10);

**Order by:**

Order by clause is used to sort the table based on any columns in ascending order or by descending order.

Order by clause executes after select clause. We can use the alias name which is given in select clause and order the data.

Syntax:

Select <\*/column name>

From <table name>

Where <filter condition>

Group by <column name>

Having <group by filter condition>

Order by <column name> asc/desc

Order of execution

1. From
2. Where
3. Group by
4. Having
5. Select
6. Order by

* Order by clause should be last statement written in the query
* Order by will always executed after select clause
* Order by clause by default sort the column in ascending order
* In order by clause we can use alias name but we cannot assign alias name.

Q. WAQTD all the employees according to date of joining only for sales man.

Select \*

From emp

Where job =sales man

Order by hiredate

**Statements**

**Statements –** they help us to create the table and insert the data.

There are 3 types of statements,

* **DDL** – Data Definition Language – the various commands in DDL are :- Create, Drop, Truncate, Alter, Rename
* **DML** – Data Manipulation Language – the various commands in DML are :- Insert, Update, Delete
* **TCL** – Transaction Control Language – the various commands in TCL are :- Rollback, Commit, Savepoint

**DDL**

**CREATE** – It creates the table.

Create table <table name>

(

<Column name> data type constraint,

.

);

Before we study the **Create** command, let us first study the some of the basic **datatypes** we use in SQL.

**1) CHAR** :-

It stores the fixed length character data.

It can store the alphanumeric data (i.e, numbers and characters).

**2) VARCHAR**

It stores the variable length character data

It can store alphanumeric data.

**Difference between CHAR & VARCHAR**

Let us consider an example as shown below to explain the difference.

***Name char (6) ;***

Here we are defining **name** which is of 6characters in length.

Now, let us store ‘***Tom***’ in the name field. Let us understand how the memory is allocated for this,

**Blank Space**

**-**

**-**

**-**

**M**

**O**

**T**

**Reserved / Non-reusable memory**

When we declare anything of type **char**, the memory is allocated as of the size given and its fixed length – hence it cannot be altered.

Now, when we give ***tom***, it allocates 6 bytes for **name char** – only the 1st 3bytes are used to store **Tom** – the rest becomes waste as it is a blank space and it is reserved memory.

The **length(name) = 6**.

**Name varchar (6) ;**

Here we are defining **name** which is of 6characters in length.

Now, let us store ‘***Tom***’ in the name field. Let us understand how the memory is allocated for this,

**Null**

**.**

**.**

**.**

**M**

**O**

**T**

**Re-usable memory**

When we declare anything of type **varchar**, the memory is allocated as shown above and it is variable length

When we give ***tom***, it allocates 6bytes for **name varchar** – only the 1st 3bytes are used to store **tom** – the remaining 3 fields becomes **null**. As we know the property of **null** – null does not occupy any memory space **– thus the memory is not wasted here.**

The **length(name) = 3**.

**Another difference is** : -

In **char**, maximum value we can store is 2000 characters

In **varchar**, maximum value we can store is 4000 characters.

**3) NUMBER**

- it stores numeric data.

**For ex – 1) sal number(4) ;**

Here the maximum possible value is 9999.

**2) sal number (6, 2)** **;**

Here, 2 – scale (total number of decimal places)

6 – precision (total number of digits including decimal places)

Maximum value is 9999.99

**sal number (4, 3) ;**

maximum value is 9.999

**sal number (2, 2)**

maximum value is .99

**4) DATE**

- it stores date and time

- no need to specify any length for this type.

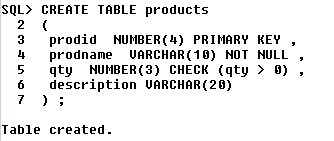
**For ex,**  SQL > order\_dt DATE ;

Date is always displayed in the default format :- **dd – month – yy**

. **Create the following tables**

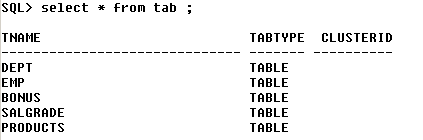
|  |  |
| --- | --- |
| **PRODUCTS** | |
| **ProdID ( PK )** |  |
| **ProdName ( Not Null )** |  |
| **Qty ( Chk > 0 )** |  |
| **Description** |  |

|  |  |
| --- | --- |
| **ORDERS** | |
| **ProdID ( FK from products )** |  |
| **OrderID ( PK )** |  |
| **Qty\_sold ( chk > 0 )** |  |
| **Price** |  |
| **Order\_Date** |  |
|  |  |

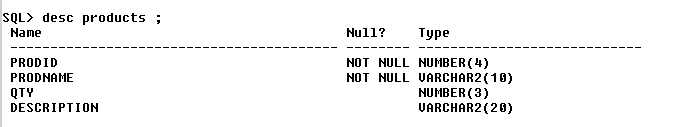


**We can see that the table has been created.**

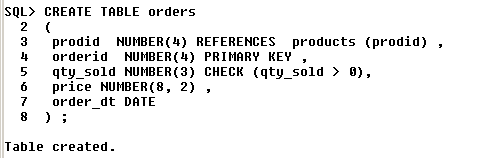
Now, let us verify if the table has really been created and also the description of the table,



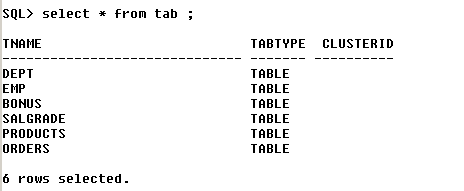
The new table **products** has been added to the database.



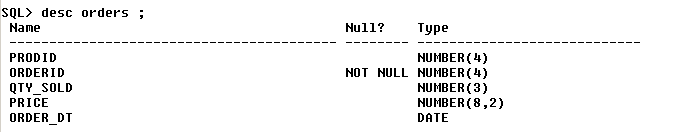
Thus, we get the description of the table **products**.



The new table **orders** has been created. We can see from the above query how to reference a child table to the parent table using the **references** keyword.



Thus we can verify that **orders** table has been created and added to the database.

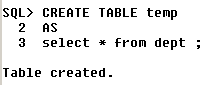


Thus, we get the description of the **orders** table.

**Creating a table from another table** :-

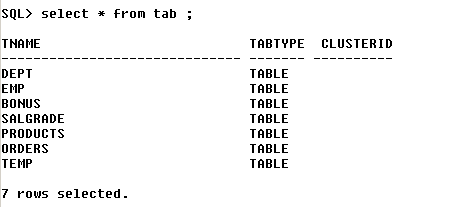
Now, we will see how to create a table from another table – i.e, it duplicates all the records and the characteristics of another table.

The SQL query for it is as follows,

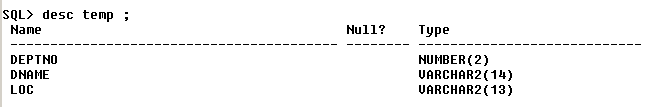


Thus we can see that we have created another table **temp** from the table **dept**.

We can verify it as shown below,



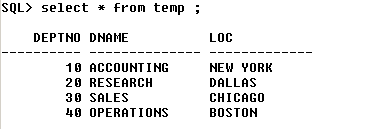
Thus, we can see that the **table temp** has been created.



Thus, we can see that the table **temp** has copied the structure of the table **dept**. Here, we must observe that **temp** copies all the columns, rows and NOT NULL constraints only from the table **dept**. It never copies PK, FK, Check constraints.

**Thus, when in the interview somebody asks you “I have a table which has about 1million records. How do I duplicate it into another table without using Insert keyword and without inserting it individually all the records into the duplicated table ?**

**Answer is - Use the above query of creating a table from another table and explain it.**



Thus, from the above query – we can see that all the records of the table **dept** has been copied into the table **temp**.

**TRUNCATE**

It removes all the data permanently, but the structure of the table remains as it is.

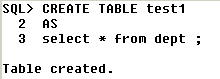
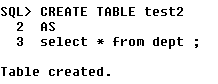
**Ex – SQL > TRUNCATE TABLE temp ;**

**DROP**

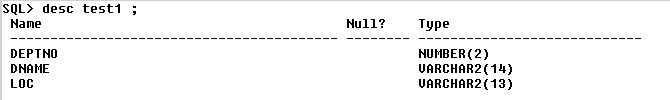
It removes both data and the structure of the table permanently from the database.

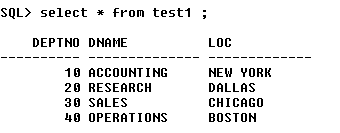
**Ex – SQL > DROP TABLE test ;**

Let us understand the difference between **drop & truncate** using the below shown example,

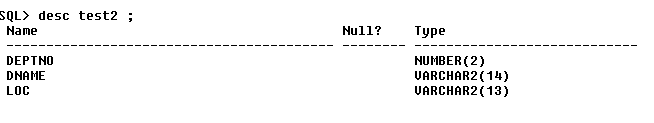
 

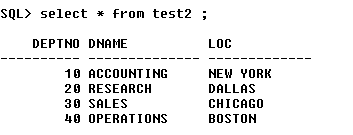
Let us create 2 tables Test1 and Test2 as shown above.





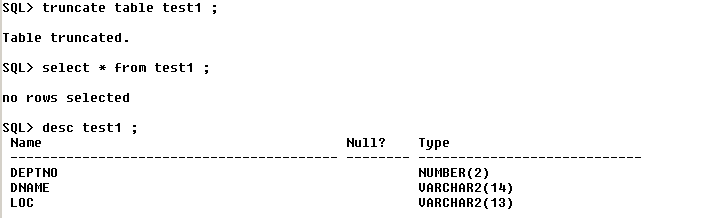
The above shows the description of the table test1.





The above gives the description of the table Test2.

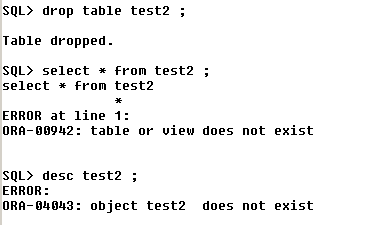
Now, let us use the **Truncate query on Test1** and **Drop query on Test2** and see the difference.



The above 3 queries show that – 1st query has the table test1 truncated.

2nd query – it shows **no rows selected** – thus only the records from the table has been removed. 3rd query – it shows that the structure of the table is still present. Only the records will be removed.

Thus, this **explains the truncate query.**



Thus from the above queries we can explain how **drop** works. 1st query – it drops the table. Thus – the entire structure and records of the table are dropped.

2nd and 3rd query – since, there is no table – **select & desc** query for **test2** will throw an error.

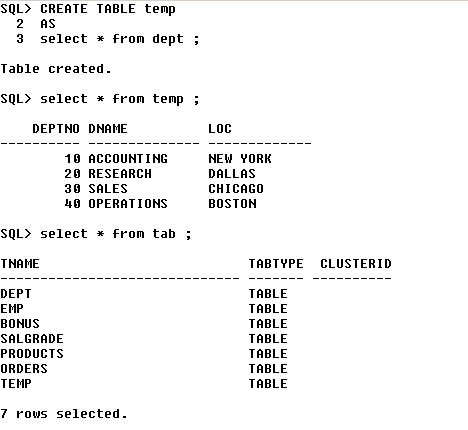
Thus, this **explains the drop query.**

Hence, we have seen the difference between **drop & truncate** query.

**RENAME**

It renames a table.

**For ex,** let us see the query of how we do this renaming a table.



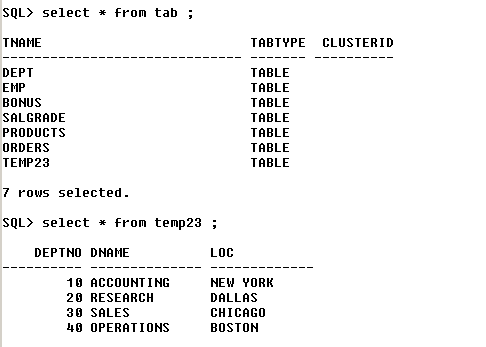
In the above 3queries – we have created a table **temp** which copies table **dept** – we see the records of the table temp – and also check if the table has really been created.

Now let us **rename temp to temp23** as shown below,



The above query is used to rename a table.

Now let us verify the contents of the table and check if it has really been modified,



Thus the table has been renamed and its contents are verified.

**ALTER**

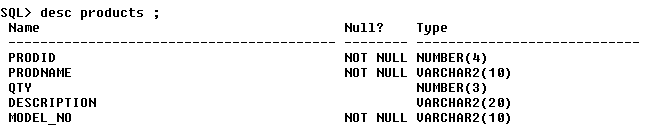
- this query alters / changes the structure of the table (i.e, - adding columns, removing columns, renaming columns etc ).

Now let us **alter** the table **products** (which we have created earlier).

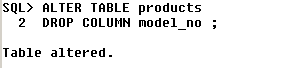
**1) Let us add a new column *‘model\_no*’ to the table.**

****

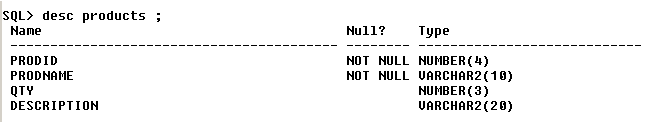
Thus, a new column has been added. Let’s verify it with the query shown below,



**2) Now let us drop the column model\_no from products.**

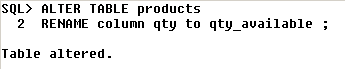


Thus, the column has been dropped.

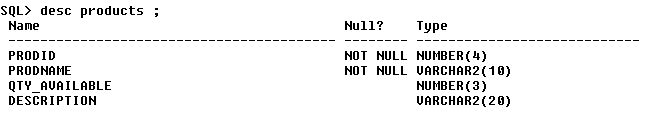


Thus, we can see from the description of the table – the column **model\_no** has been dropped.

**3) Let us rename the column *qty* to *qty\_available***.



Let us verify if it has been renamed,



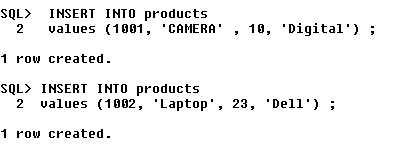
**NOTE : *SELECT* is neither DML nor DDL. It does not belong to any group because it does not alter anything, it just displays the data as required by the user**.

**DML**

**INSERT**

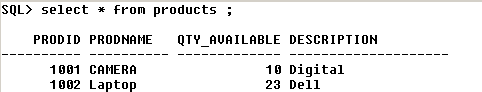
It inserts a record to a table.

Let us observe how it is done,



This is how we insert values into a table. All characters and alpha-numeric characters(ex – 10023sdf78) must be enclosed in single quotes (‘ ‘ ) and each value must be separated by comma. Also we must be careful in entering the data without violating the primary key, foreign key , unique constraints.

Now let us see the table in which the data in has been inserted,

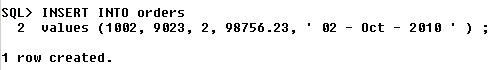


Now, let us insert data into the table **orders** in which a foreign key is referencing primary key,

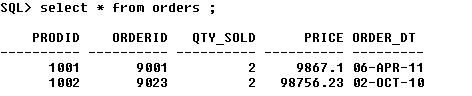


Here, we see that 1001 is the same prodid as of the earlier table.

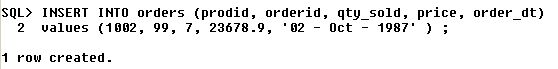
Sysdate – it displays the current date set in the system .



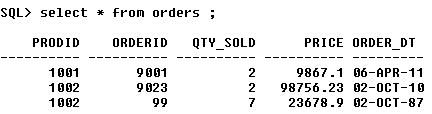
Now, let us see the table,



Another way of inserting data into the table is shown below,



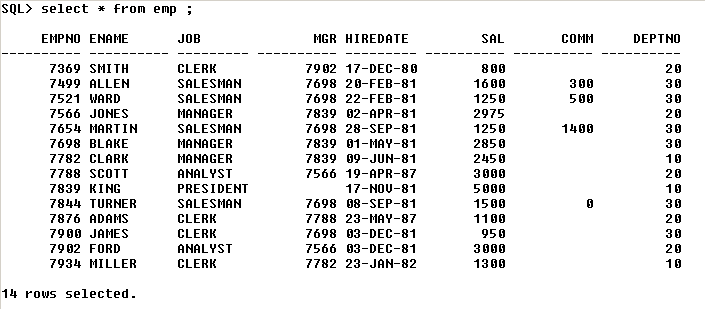
Now, let us see the table,



**UPDATE** :-

It updates one or more records.

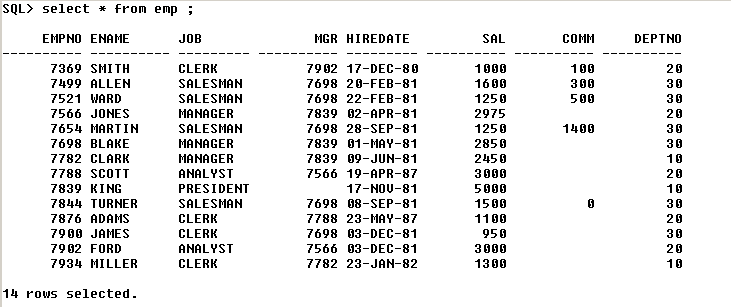
**For ex – 1)** Let us update salary by increasing it by Rs200 and also give commission of Rs100 where empno = 7369.



Now, let us **update** the said record as shown below,



Let us verify if the record has been updated,

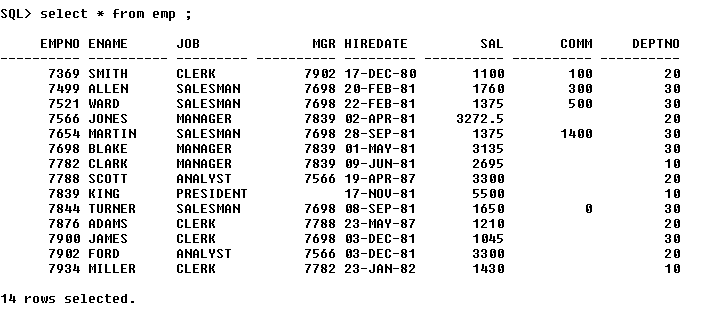


Thus, the record(empno – 7369) has been updated.

**2) Increase all salary by 10%**



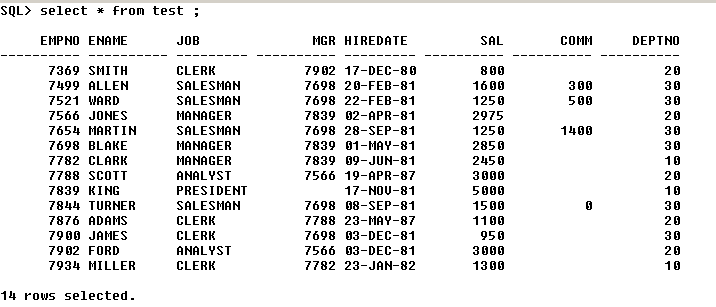
Let us verify it,



**DELETE**

It deletes one / some / all the records.

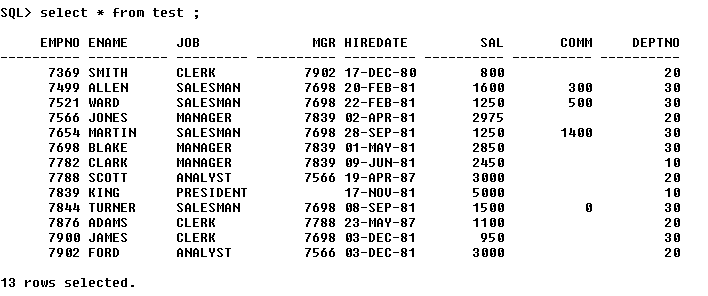
Let us create a table test from table emp – and see how to delete 1 record and how to delete all records from it,



Thus, we have created the table test.



Thus 1 row, ‘miller’ has been deleted.



Thus, the deletion has been confirmed.

**TCL**

Any DML change on a table is not a permanent one.

We need to save the DML changes in order to make it permanent

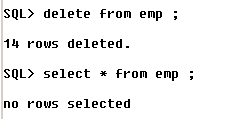
We can also undo (ignore) the same DML changes on a table.

The DDL changes cannot be undone as they are implicitly saved.

**ROLLBACK**

It undoes the DML changes performed on a table.

Let us see in the below example how **rollback** works,

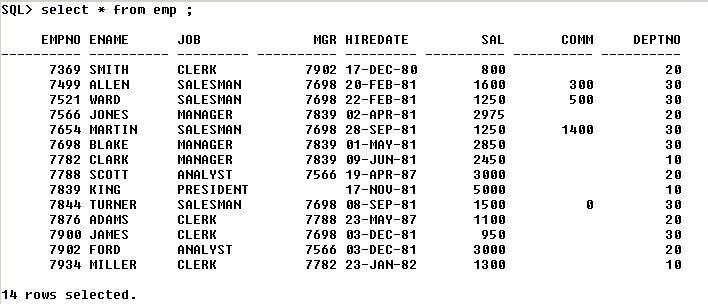


Let us delete the employee table. When we perform **select** operation on emp, we can see that all the rows have been deleted.

We now perform the **rollback** operation,



Now let us perform the **select** operation,



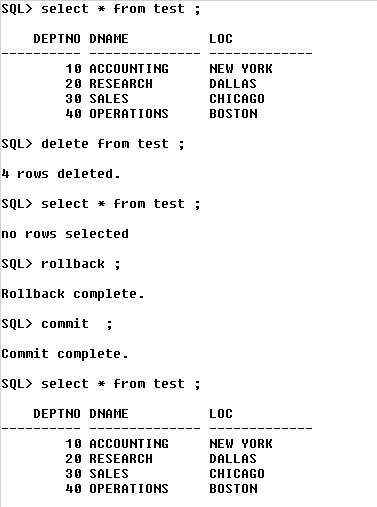
Thus performing the **rollback** operation, we can retrieve all the records which had been deleted.

**COMMIT**

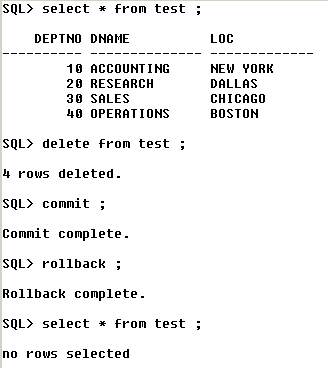
It saves the DML changes permanently to the database.

**Committing after rollback & vice versa will not have any effect**

Let us explain the above statement with an example,



We can see that **commit** has no effect after **rollback** operation.



Thus, from above – we can see that **rollback** has no effect after **commit** operation.

During an abnormal exit – i.e, shutdown or if the SQL window is closed by mouse click – then all the DML’s will be rolled back automatically.

During a normal exit – **exit ;** - all the DML’s will be auto-committed – and there will be no rollback.

**Ex – 1)** INSERT

UPDATE

ALTER

DELETE

ROLLBACK

When we perform the following operations in the same order for a table – then INSERT, UPDATE will be committed – because ALTER is a DDL – and thus all the DML’s above it will also be committed – because DDL operations cannot be undone.

Here – only DELETE will be rolled back because it’s a DML.

**2)** INSERT

UPDATE

DELETE

ROLLBACK

Here, all are rolled back.

**SAVEPOINT** :

It is like a pointer (break-point) till where a DML will be rolled back.

**Ex :-**

Insert …

Save point x ;

Update …

Delete ..

Rollback to x ;

…

…

Here, only DELETE & UPDATE are rolled back.

INSERT is neither rolled back nor committed.

**Assignments**

**1) Create the following tables**

a) Table name :- STUDENTS

regno (PK)

name (NN)

semester

DOB

Phone

b) Table name :- BOOKS

bookno (PK)

bname

author

c) Table name :- LIBRARY

regno (FK from students)

bookno (FK from books)

DOI –date of issue

DOR – date of return

**2) Insert 5 records to each of these tables**

**3) Differentiate between,**

**a) Delete and Truncate**

**b) Truncate and Drop**

**c) Char and Varchar**

**d) Drop and Delete**

**Single row functions**

Functions – it is a re-usable program that returns a value.

Single row functions executes row by row that is it provide output for every record given as input.

Input argument of a single row function can be column name or expression.

* GROUP functions
* CHARACTER functions
* NUMERIC functions
* DATE functions
* SPECIAL functions

We have already learnt about GROUP functions.

Now, let us study the various CHARACTER functions.

**CHARACTER functions**

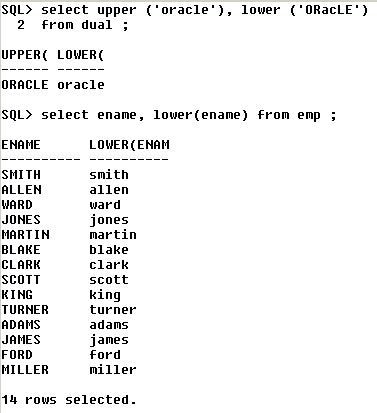
a) Upper: - it is used to convert the given string to upper case

b) Lower: - it is used to convert the given string to lower case

c) Length: - it is used to obtain no. of characters or digits present in the given string or no.

d)initcap: - it is used to convert the given string into init cap case.

**For ex :-**

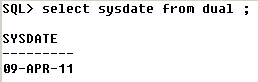


In the 1st query, we see something called as **dual**.

**Dual –** is a dummy table which is used for performing some independent operations which will not depend on any of the existing tables.

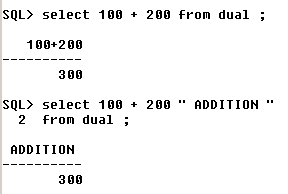
**For ex,**

1)

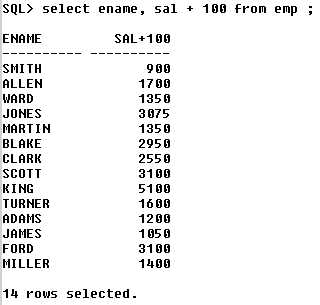


This gives the system date.

2)



3)

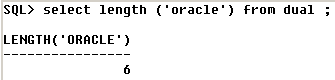


We use dual – when the data is not present in any of the existing tables. Then we use dual.

**Length** – it returns the length of a given string.

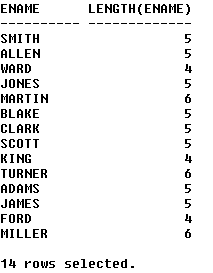
**For ex,**

**1)**



**2)**





1. **Display all the employees whose name & job is having exactly 5 characters**

**Select \***

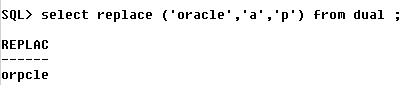
**From emp**

**Where length(ename ) and length(job) =5;**

**REPLACE**

It replaces the old value with a new value in the given string.

**For ex,**

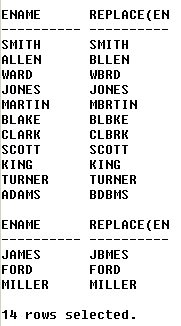


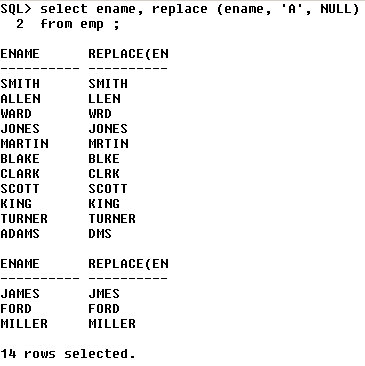
Here, **a –** is the old value to be replaced with **p** – which is the new value.



This query replaces all the names which has ‘A’ in it with ‘B’.

Let us see the output as shown below,



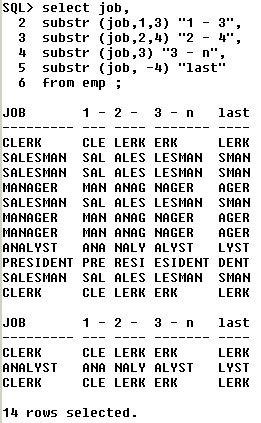


**SUBSTR**

Substring function is used to obtain a new string from a given string. This is called **substring**.

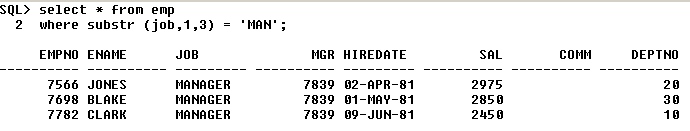
It extracts ‘n’ characters from x(th) position of a given string.

**For ex,**



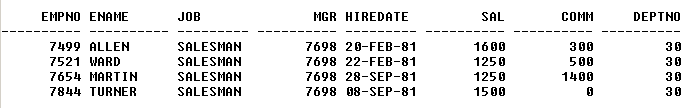
Here **, (job, ‘1’ , ‘3’)** – means from **job** – extract **from 1st position , 3 characters**.

**1) Display the employees whose job starts with ‘man’**

****

**2) Display the employees whose job ends with ‘man’**

****

****

**INSTR**

This is also called as **instring.**

It returns position of a given character in a given string.

**For ex,**

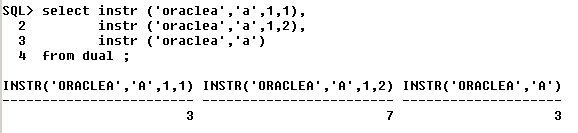
***Select instr (‘oracle’ , ‘a’ , 1 , 1) from dual ;***

Given string

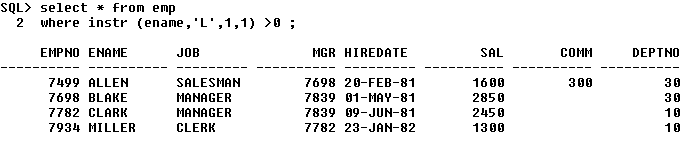
Character to be searched

Position from where the search should begin

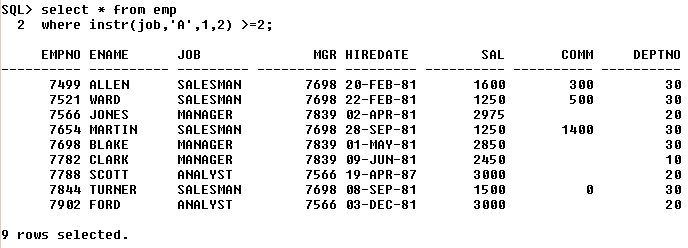
Number of occurences



**Display all the employees whose name is having ‘L’**



**List the employees whose job is having atleast 2 A’s in it**

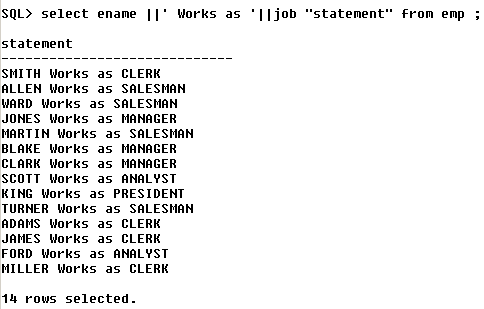


**CONCAT**

It concatenates any two values or columns.

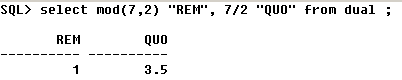
It is represented by - **||**

**For ex,**

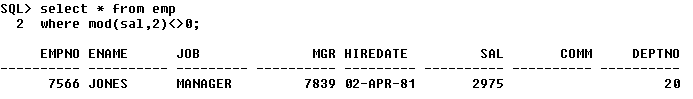


**NUMERIC FUNCTIONS**

1) **Mod** :- it returns the remainder when 1 number is divided by the other.



**Display the employees earning odd numbered salaries.**

****

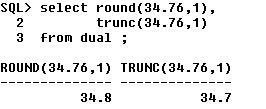
**Round**

It rounds off a given number to the nearest decimal place.

**Trunc**

It truncates the given number to the given decimal place. Truncate does not do any rounding.

**For ex,**



Here, ‘**1**’ indicates the number of positions.

**DATE FUNCTIONS**

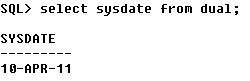
**1) Sysdate**

Stands for System date.

It returns both date & time, but by default – only date is displayed.

The default format is,

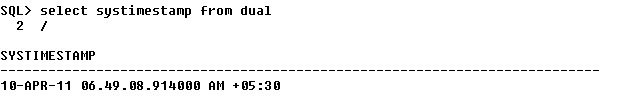
**dd – mon – yy**

****

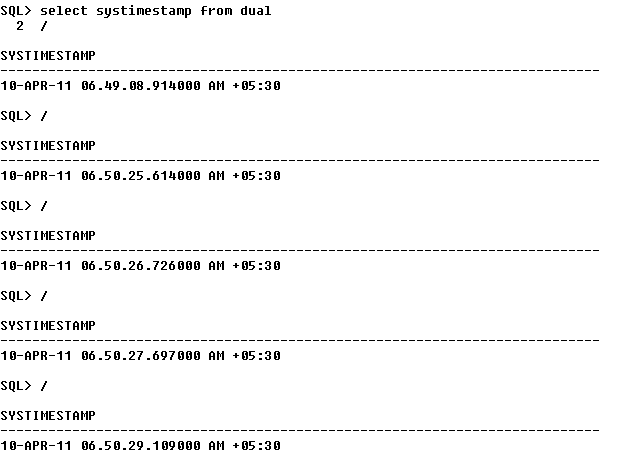
**2) Systimestamp**

Introduced from Oracle 9i

Returns date, time and timezone.

****

Here, **.914000** – gives the fraction of millisecond which keeps changing as shown below,

****

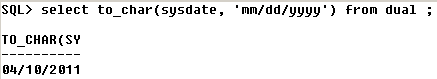
**In interview – if they ask you – *“ which function contains fractions of a second “ OR “how to see the system time “* – then answer is “SYSTIMESTAMP”.**

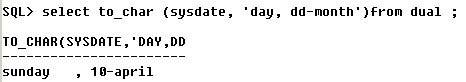
**SPECIAL FUNCTIONS**

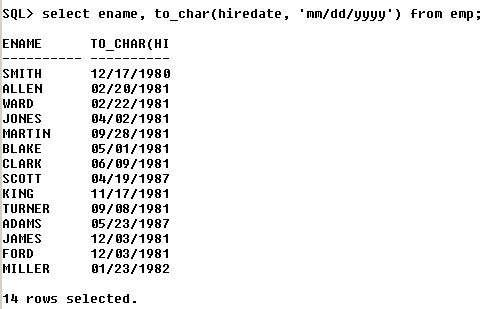
**1) TO – CHAR**

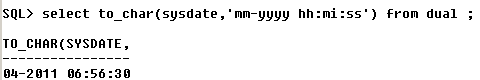
Used for displaying the date in different formats.

**For ex,**

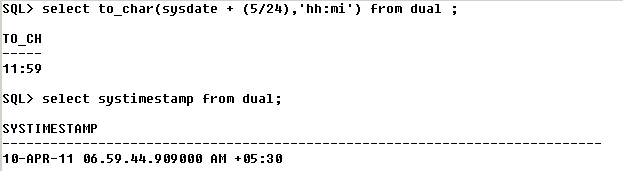
****

****

****

****

**Now, let us see how to add 5 hrs to the existing time,**

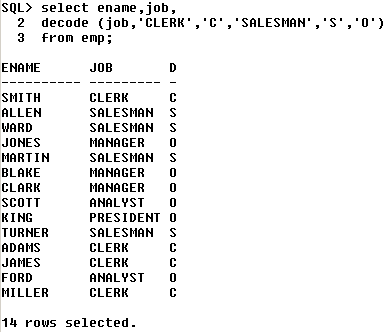
****

We can see that 5 hrs has been added to the current time.

**DECODE**

It works like ‘**if – then – else**’ statement.

**For ex,**

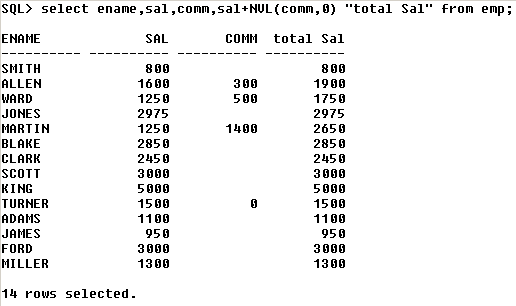


The above query states that – in job, if clerk is there, replace with C – else if salesman is there, replace it with S – else replace with ‘O’.

**NVL**

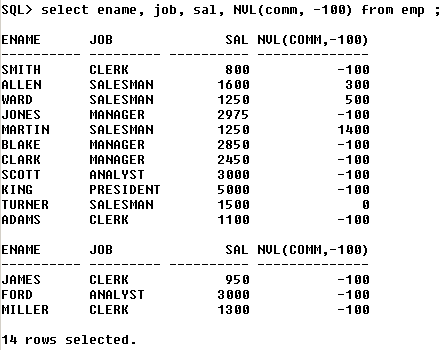
It substitutes a value for a null.

**For ex,**

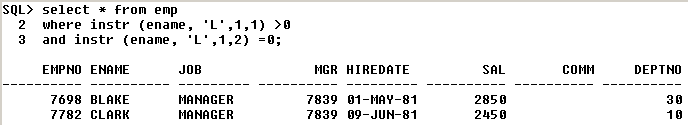
****

The above query means – if the employee has commission, then add sal + comm. To get total salary – else add 0 to the sal and display total salary.

**Display employee name, job, salary and commission. If the commission is NULL, then display -100**



**Display all employees whose name is having exactly 1 ‘L’ in it**

****