***CHAPTER 1***

**RDBMS Concepts**

**Database**

A database is the place of storage of the data in the form of tables

Data means information which is very useful. A database is also collection of 1 or more tables.

**Table** – a table is a collection of rows and columns.

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

**Columns**

A cell is an intersection of a row and a column

A column is also called as a field / attribute

A record is also called as a row / tuple.

A table is also called as an entity / relation.

***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
* A database can also contain other database objects like views, indexes, stored procedures, functions, triggers etc, apart from tables.

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

**CONSTRAINTS**

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

**NULL**

* NULL is nothing, it is neither zero nor blank 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 atleast 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** | |

***CHAPTER 2***

**SQL – Structured Query Language**

SQL – Structured Query Language

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

**Troubleshooting Oracle**

***Error 1***

The account is locked

***Steps to rectify the error***

* Login as username – ‘system’ & password – ‘manager’ or ‘password – ‘tiger’
* SQL > show user ;

User is “SYSTEM”

SQL > alter user scott account unlock ;

User altered

SQL > exit ;

***Error 2***

TNS : protocol adapter error

***How to troubleshoot this***

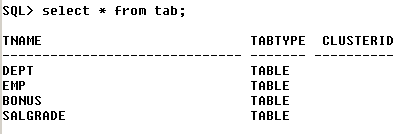
Cause is “oracle service has not started”

How to go here,

Settings – Control Panel – Administrative Tools – Services

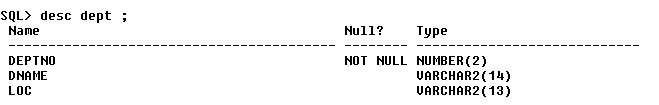
Sort the list

There is an “Oracle Service ORCL” & “start the service”

****

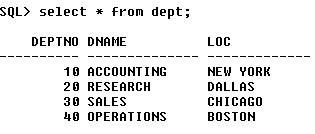
This query gives the list of tables.

\* - selects all

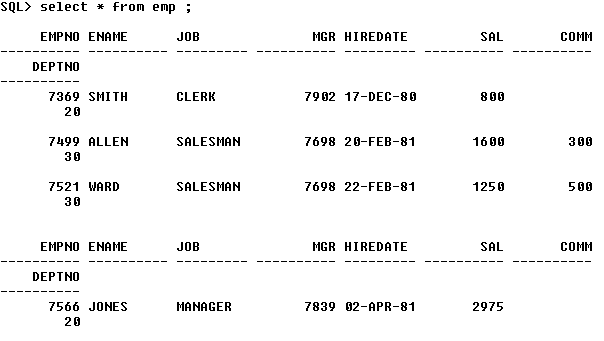


This query gives the description of the table “department”.

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

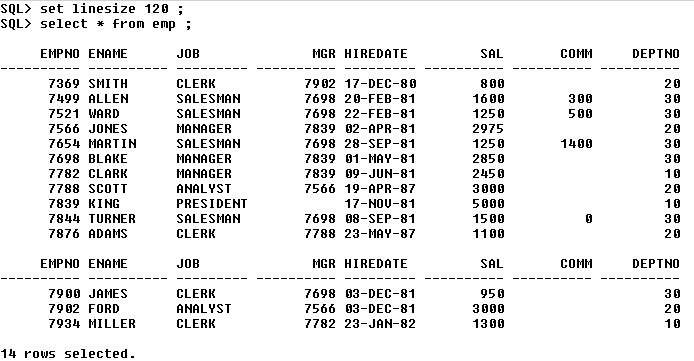


This query gives the description of the table “department”

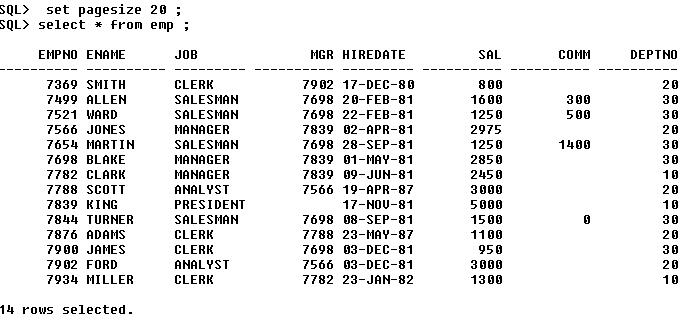


The above query gives the description of the “employee” table. But we see that all the data is in different lines which makes it very difficult to analyse.

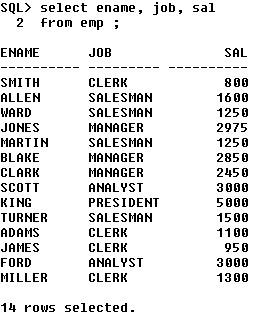
So we use the following command to see the data in a more orderly fashion,



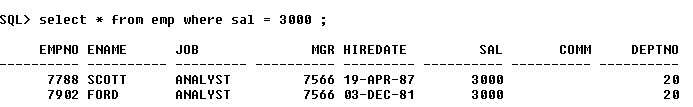
The **“set linesize”** command helps in increasing the line size , thus the data is arranged in a orderly fashion.



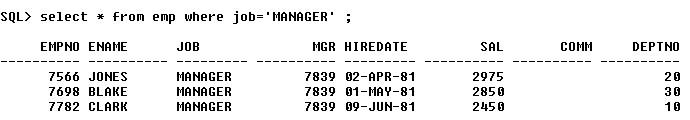
The above command **“set pagesize 20”** increases the page size, thus accommodating more number of rows in a single page.



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



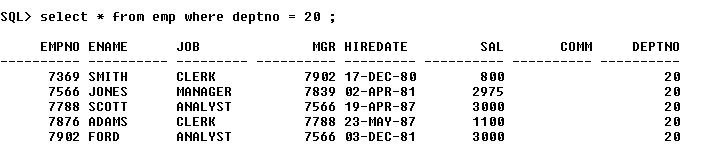
**‘where’** clause is used to restrict the number of records displayed. It gives only the records of the specified condition.



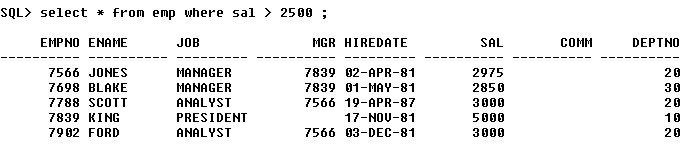
Any string data should be enclosed within **single quotes** ( ‘ ‘ ) and the same becomes **case sensitive**.

**Assignment**

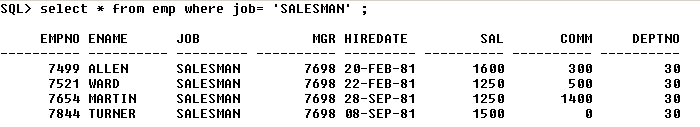
**1) List the employees in dept 20**

****

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



**3) Display all salesmen**



***CHAPTER 3***

**OPERATORS**

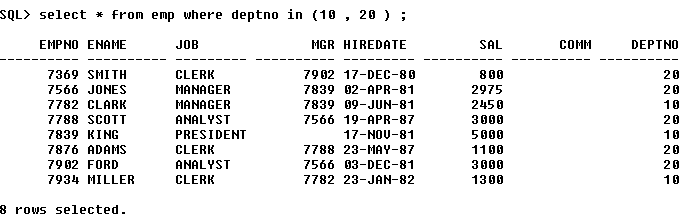
Operators are classified into,

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

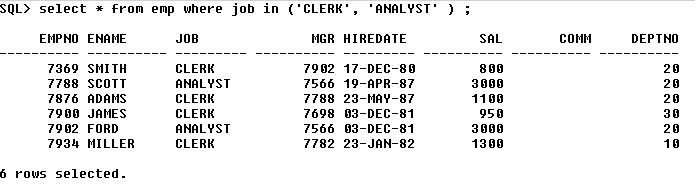
**SPECIAL OPERATORS**

**1) IN** – it is used for evaluating multiple values.

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



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



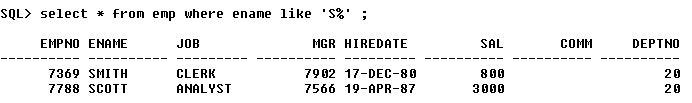
We can provide upto 1000 values at the max

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

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

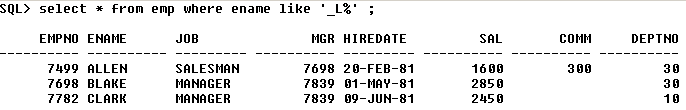
**\_ (underscore)** - matches exactly one character

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



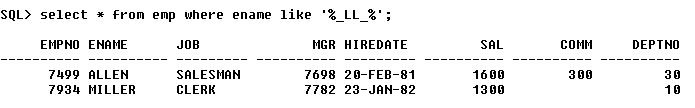
Whenever we use % or \_ , always ensure that it is preceded by the word **‘like’**

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

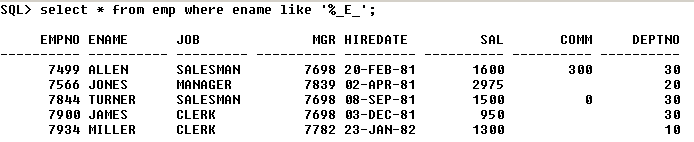


**ASSIGNMENT**

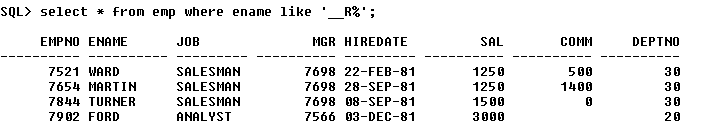
**1) List the employees whose name is having atleast 2 L’s**



**2) List the employees whose name is having letter ‘E’ as the last but one character**

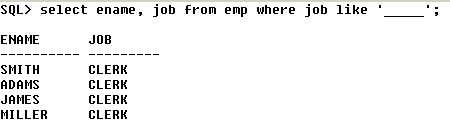
****

**3) List all the employees whose name is having letter ‘R’ in the 3rd position**



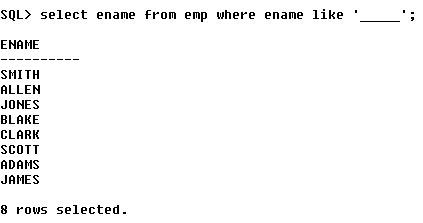
In the above query, we give 2 underscores before R%.

**4) List all the employees who are having exactly 5 characters in their jobs**



Here , in single quotes – we give 5 underscores.

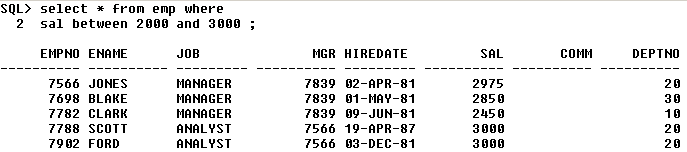
**5) List the employees whose name is having atleast 5 characters**



Here, also in single quotes – we give 5 underscores ( \_\_\_\_\_ )

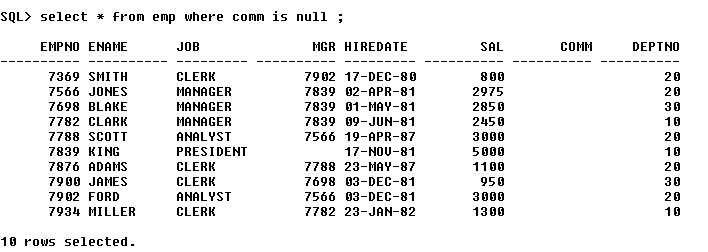
**3) BETWEEN** operator – used for searching based on range of values.

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



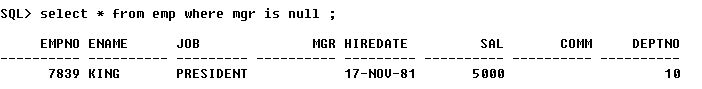
**4) IS** operator – it is used to compare nulls

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



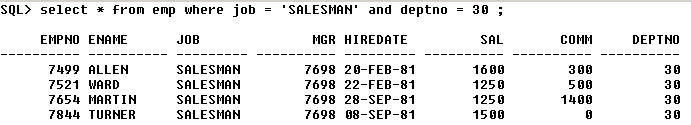
**ASSIGNMENT**

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

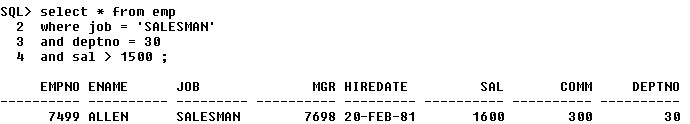


**LOGICAL OPERATORS**

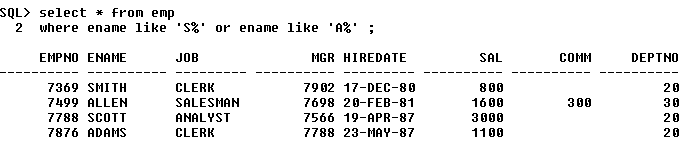
**1) List all the salesmen in dept 30**

****

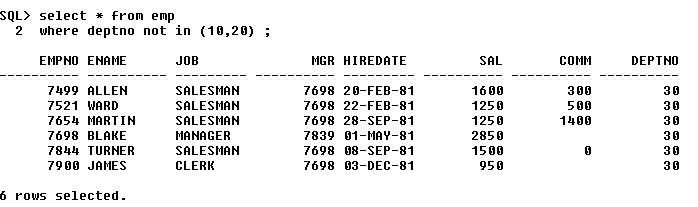
**2) List all the salesmen in dept number 30 and having salary greater than 1500**

****

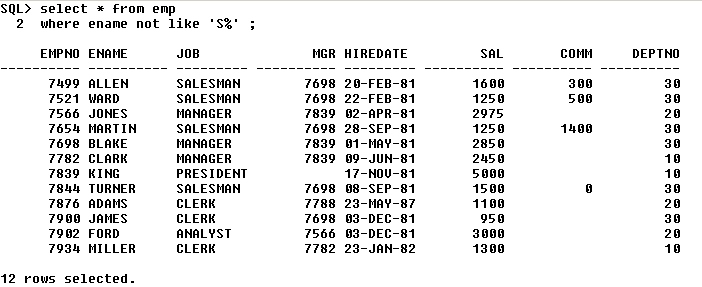
**3) List all the employees whose name starts with ‘s’ or ‘a’**

****

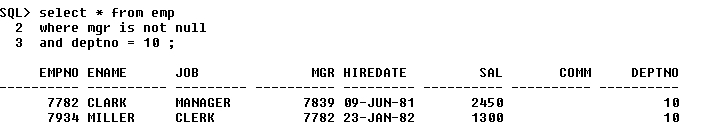
**4) List all the employees except those who are working in dept 10 & 20.**

****

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

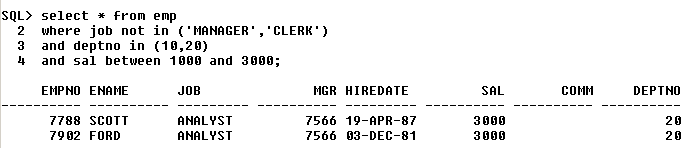
****

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

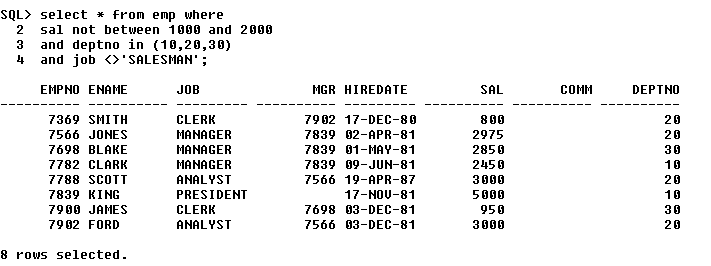
****

**ASSIGNMENT**

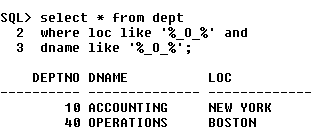
**1) List the employees who are not working as managers and clerks in dept 10 and 20 with a salary in the range of 1000 to 3000**

****

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

****

**3) List the department names which are having letter ‘O’ in their locations as well as their department names**

****

**SORTING**

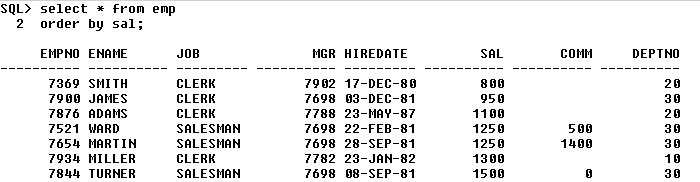
It arranges the data either in ascending / descending order

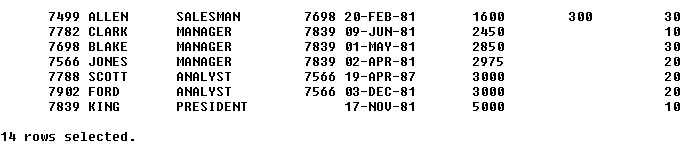
Ascending – ASC / Descending – DESC

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

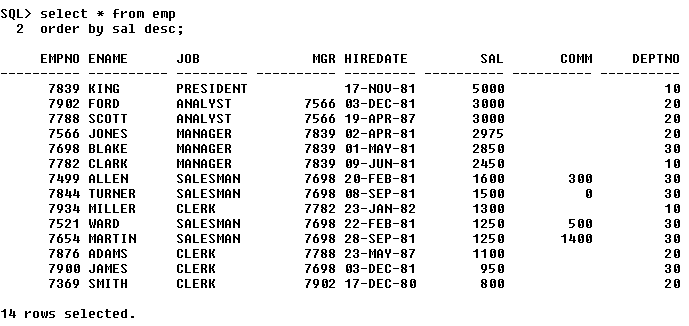
By default, the data is always arranged in ASC order

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

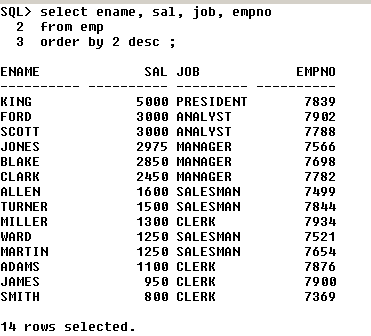


****

**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**

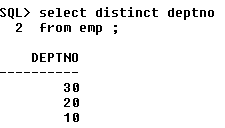
****

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.

**Selecting DISTINCT VALUES**



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

***CHAPTER 4***

**GROUP functions and Grouping**

We have **5 GROUP** functions,

1. Sum
2. Max
3. Min
4. Avg
5. Count

**Sum –** returns total value

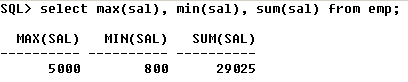
**Max** – returns maximum value

**Min** – returns minimum value

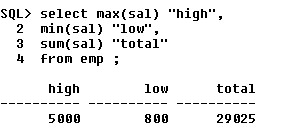
**Avg** – returns average value

**Count –** returns number of records

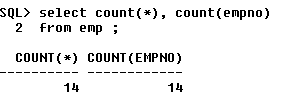
**Ex – 1) display the maximum salary, minimum salary and total salary from employee**



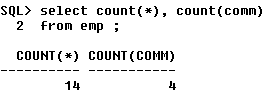
***To give aliases for the columns*** *:-*



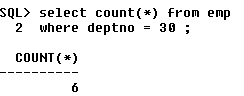
**3) The below query gives the total number of employees**



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

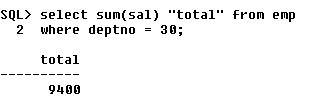
****

**5) List the number of employees in department 30**

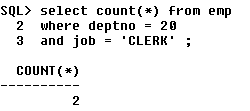
****

**ASSIGMENT**

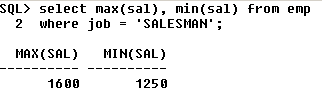
**1) Display the total salary in department 30**



**2) List the number of clerks in department 20**

****

**3) List the highest and lowest salary earned by salesmen**

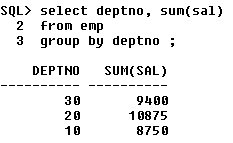
****

**GROUPING**

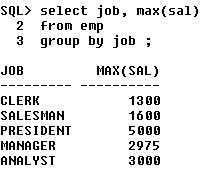
It is the process of computing the aggregates by segregating based on one or more columns.

Grouping is done by using **‘group by’** clause.

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

****

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

****

**HAVING**

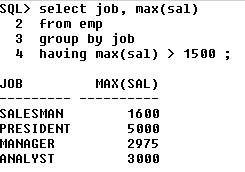
**‘Having’** is used to filter the grouped data.

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

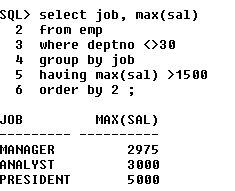
‘**Having’** should be used after **group by** clause

**‘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**

****

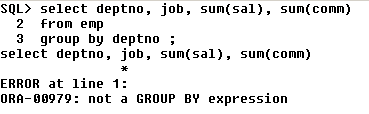
**2) Display job-wise highest salary only if the highest salary is more than 1500 excluding department 30. Sort the data based on highest salary in the ascending order.**

****

**RESTRICTIONS ON GROUPING**

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

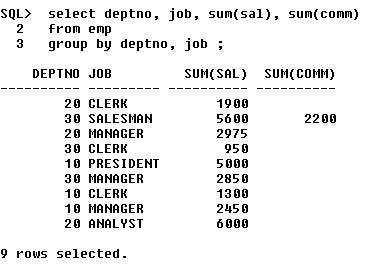
If we try selecting other columns, we will get an error as shown below,

****

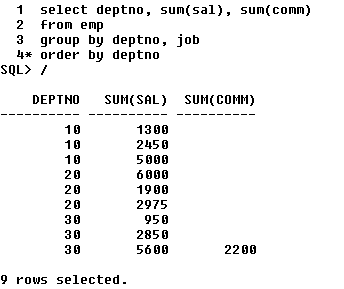
The above query is an error because ‘**job**’ is there in the **select** query but not in the **group by** query.

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,



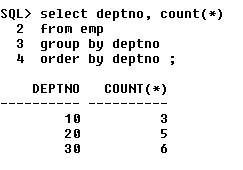
The below query is also correct to rectify the above error,



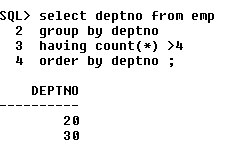
Whatever is there in the **select** statement must be there in the **group by** statement. But, whatever is there in the **group by** statement need not be present in the **select** statement. This is shown in the above two corrected queries.

**ASSIGNMENT**

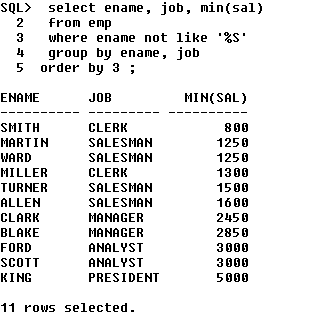
**1) Display the department numbers along with the number of employees in it**



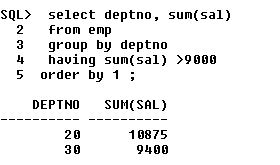
**2) Display the department numbers which are having more than 4 employees in them**

****

**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**

****

**NOTE :**

To clear the screen, the command used is,

**cl scr ;**

if it is a large query and we cannot type it repeatedly, then type in – **SQL > ed ;**

when we type **ed ;** - we get the notepad – after making the necessary changes – then click on the **‘x’** i.e, the close button at the top right corner – then click on **yes** when a dialog box asking whether to overwrite the file comes – after this it comes to the oracle screen – in the next line , enter ‘**/** ‘ and hit on **enter** button – another way of ending the query is by typing ‘ **/** ‘ in the next line of the query – this indicates the end of the query.

***CHAPTER 5***

**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

**CREATE** – It creates the table.

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 6 characters 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**

**NOTE :-**

**varchar2** – from 10g, varchar & varchar2 are the same.

Earlier, varchar was supporting upto 2000 characters and varchar2 was supporting upto 4000 characters.

**5) BLOB**

Stands for – Binary Large Object

It stores binary data (images, movies, music files) within the database. It stores upto 4GB.

**6) CLOB**

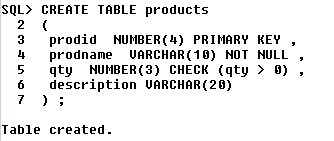
Stands for – Character Large Object

It stores plain character data like **varchar** field upto 4GB.

**Create the following tables**

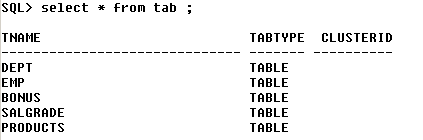
|  |  |
| --- | --- |
| **PRODUCTS** | |
| **ProdID ( PK )** |  |
| **ProdName ( Not Null )** |  |
| **Qty ( Chk > 0 )** |  |
| **Description** |  |

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

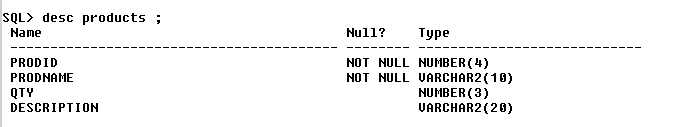


**We can see that the table has been created.**

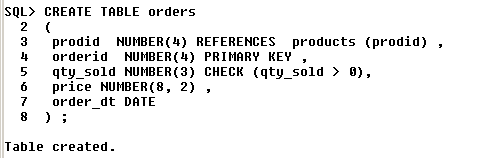
Now, let us verify if the table has really been created and also the description of the table,



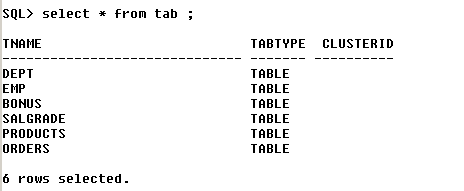
The new table **products** has been added to the database.



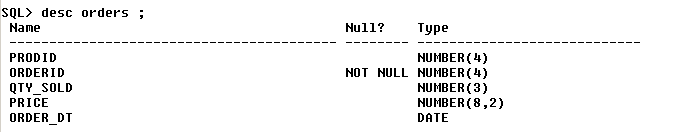
Thus, we get the description of the table **products**.



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 ben created and added to the database.

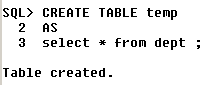


Thus, we get the description of the **orders** table.

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

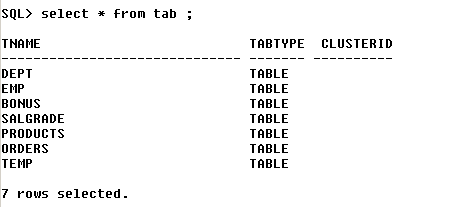
Now, we will see how to create a table from another table – i.e, it duplicates all the records and the characterstics of another table.

The SQL query for it is as follows,

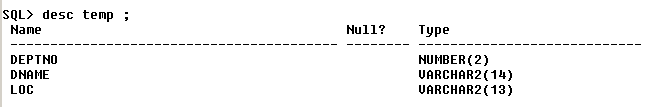


Thus we can see that we have created another table **temp** from the table **dept**.

We can verify it as shown below,



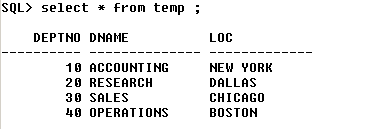
Thus, we can see that the **table temp** has been created.



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.

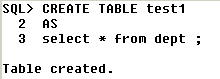
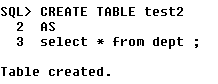
**Ex – SQL > TRUNCATE TABLE test ;**

**DROP**

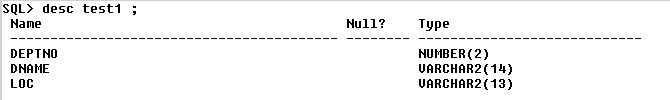
It removes both data and the structure of the table permanently from the database.

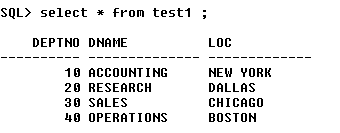
**Ex – SQL > DROP TABLE test ;**

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

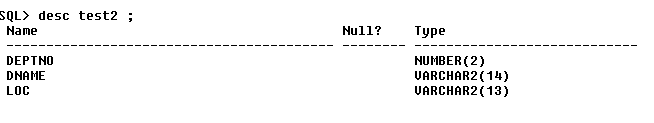
 

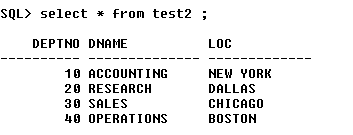
Let us create 2 tables Test1 and Test2 as shown above.





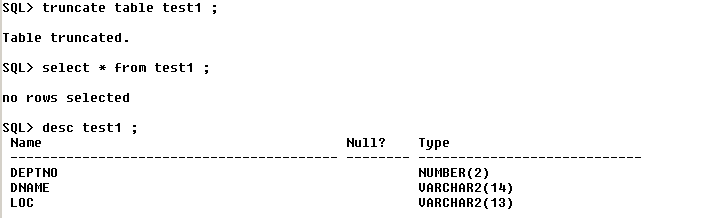
The above shows the description of the table test1.





The above gives the description of the table Test2.

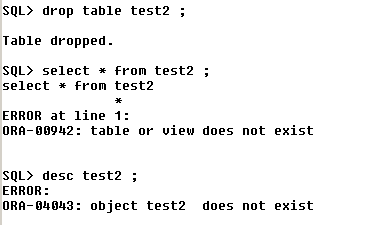
Now, let us use the **Truncate query on Test1** and **Drop query on Test2** and see the difference.



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.

**10g Recycle Bin**

**TABLE**

**DROP**

**RESTORE IT**

**PERMANENTLY REMOVE**

**RECYCLE**

**BIN**

**What to do?**

**FLASHBACK**

**PURGE**

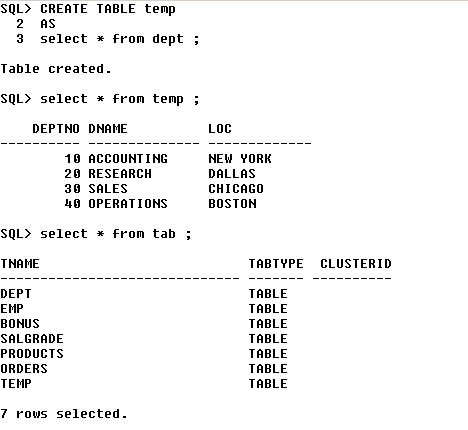
The functionality of Recycle Bin was introduced in Oracle 10G version only. Thus even though the table has been dropped, we can still restore it using **flashback command** or we can permanently remove it using the **purge** command.

This concept of Recycle bin was not there in the earlier versions of Oracle.

**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.

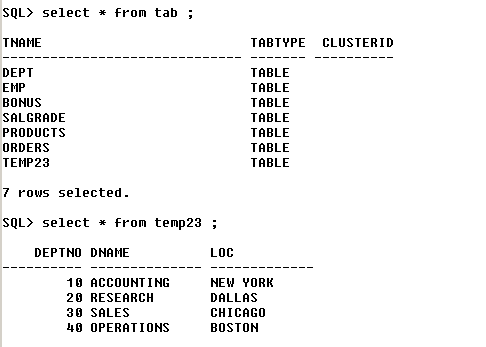
Now let us **rename temp to temp23** as shown below,



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,

See next page,



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

**ALTER**

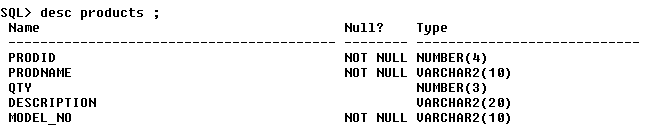
- this query alters / changes the structure of the table (i.e, - adding columns, removing columns, renaming columns etc ).

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

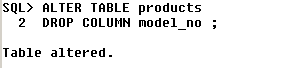
**1) Let us add a new column *‘model\_no*’ to the table.**

****

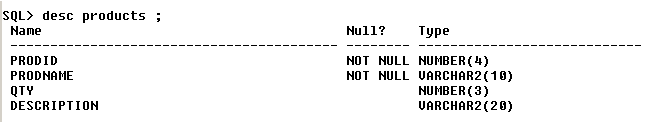
Thus, a new column has been added. Lets verify it with the query shown below,



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

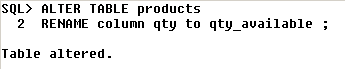


Thus, the column has been dropped.

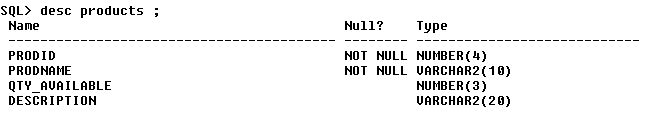


Thus, we can see from the description of the table – the column **model\_no** has been dropped.

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



Let us verify if it has been renamed,



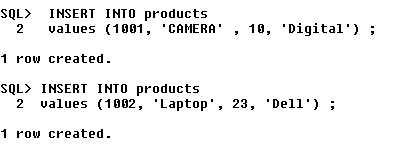
**NOTE : *SELECT* is neither DML nor DDL. It does not belong to any group because it does not alter anything, it just displays the data as required by the user**.

**DML**

**INSERT**

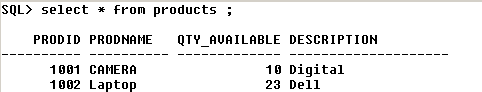
It inserts a record to a table.

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.

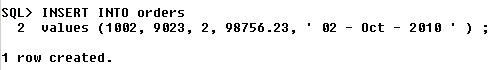
Now let us see the table in which the data in has been inserted,



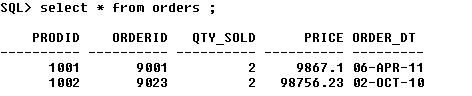
Now, let us insert data into the table **orders** in which a foreign key is referencing primary key,



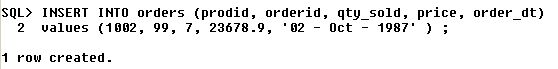
Here, we see that 1001 is the same prodid as of the earlier table. Sysdate – it displays the current date set in the system .



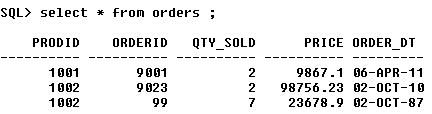
Now, let us see the table,



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



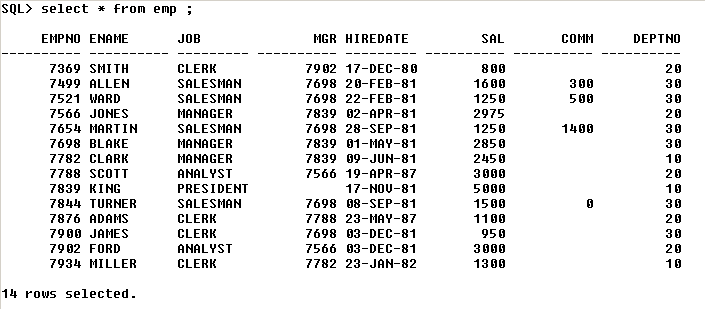
Now, let us see the table,



**UPDATE** :-

It updates one or more records.

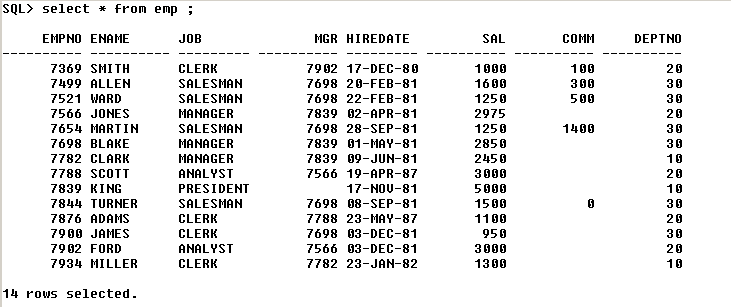
**For ex – 1)** Let us update salary by increasing it by Rs200 and also give commission of Rs100 where empno = 7369.



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



Let us verify if the record has been updated,

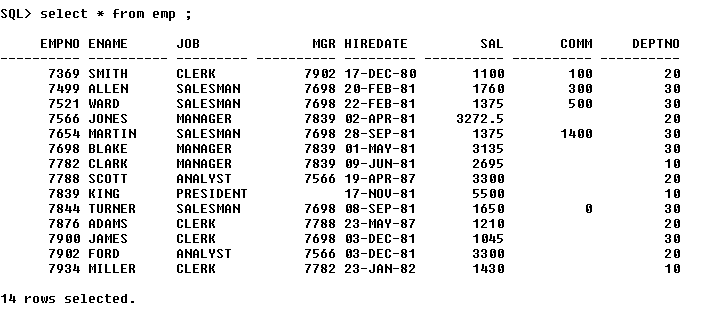


Thus, the record(empno – 7369) has been updated.

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



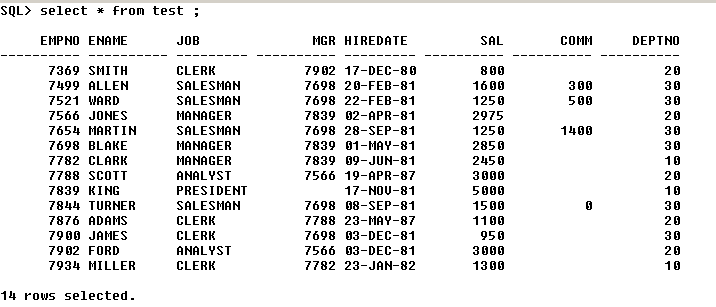
Let us verify it,



**DELETE**

It deletes one / some / all the records.

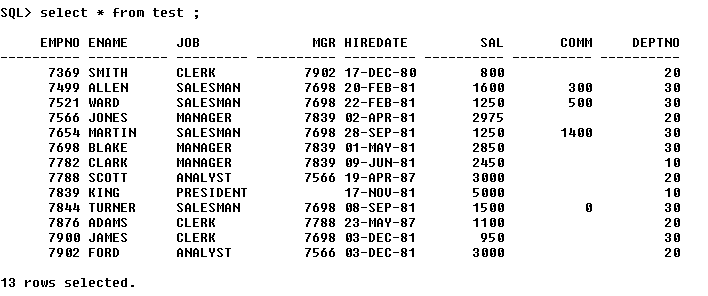
Let us create a table test from table emp – and see how to delete 1 record and how to delete all records from it,



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

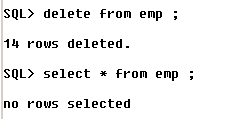
We can also undo (ignore) the same DML changes on a table.

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,

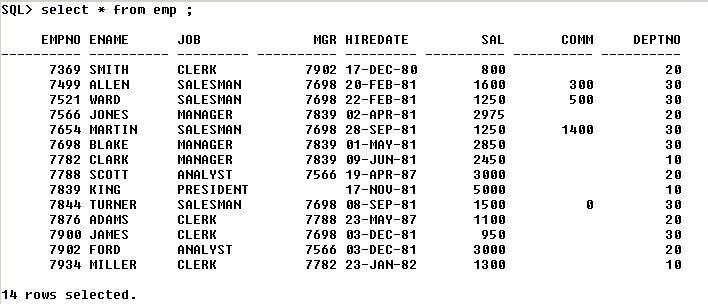


Let us delete the employee table. When we perform **select** operation on emp, we can see that all the rows have been deleted.

We now perform the **rollback** operation,



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



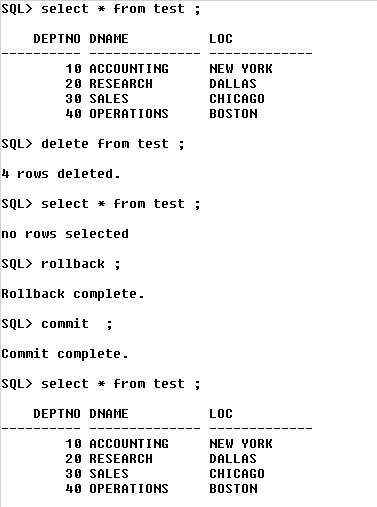
Thus performing the **rollback** operation, we can retrieve all the records which had been deleted.

**COMMIT**

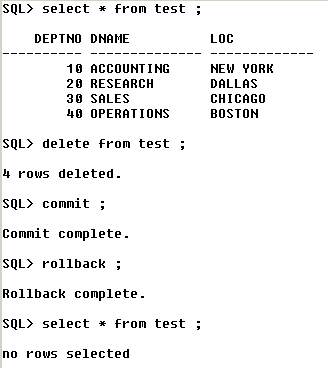
It saves the DML changes permanently to the database.

**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**

Delete and Truncate

a) Delete – deletes whichever records we want to delete from the table

Truncate – deletes all the records whether we want it or not

b) Delete – can be undone

Truncate – cannot be undone.

**NOTE** – The Primary Key created using more than 1 column is called as ***composite primary key***.

Ex – **alter table lib**

**Add primary key (regno, bookno, DOI)** ;

***CHAPTER 6***

**SUB - QUERIES**

A sub-query is also called as a nested query.

**Syntax of a sub-query**

**Select …**

**From …**

**Where … ( select …**

**From …**

**Where …**

**)**

**OUTER QUERY INNER QUERY**

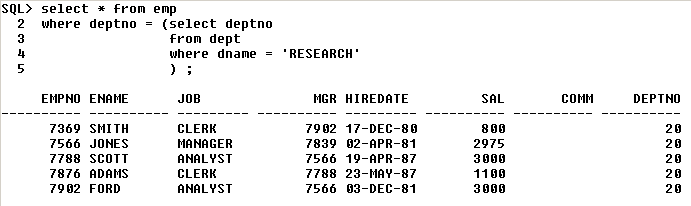
Here, the **inner query** will be executed first.

The output of **inner query** is passed as input to the **outer query.**

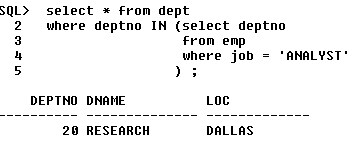
To write a sub-query, atleast 1 common column should be existing between the tables.

**For ex :-**

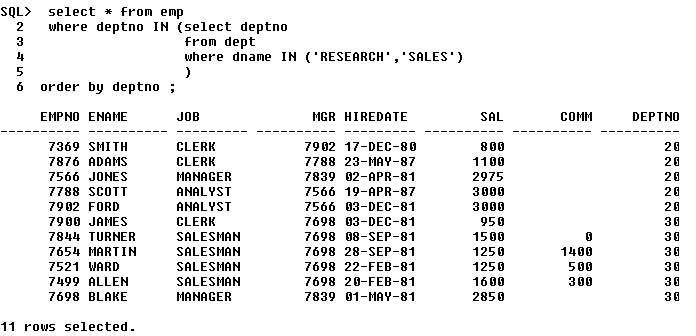
**1) List the employees working in ‘Research’ department.**



**2) List the department names that are having analysts**

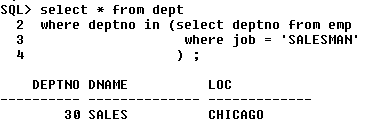
****

**3) List the employees in Research and Sales department**

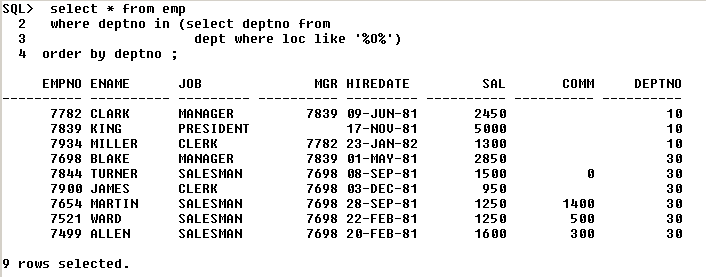
****

**Assignment**

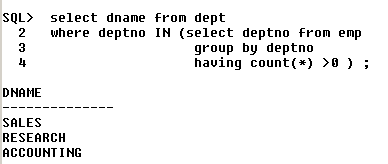
**1) List the department names which are having salesmen in it.**

****

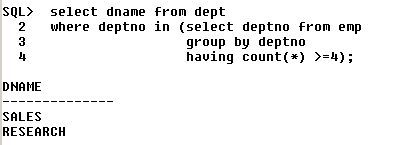
**2) Display the employees whose location is having atleast one ‘O’ in it.**

****

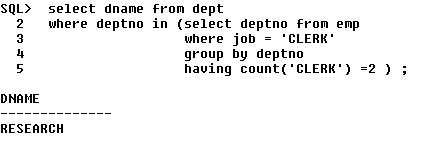
**3) List the department names that are having atleast 1 employee in it.**



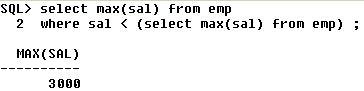
**4) List the department names that are having atleast 4 employees in it**



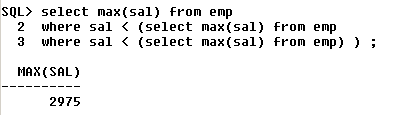
**5) Display the department names which are having atleast 2clerks in it**



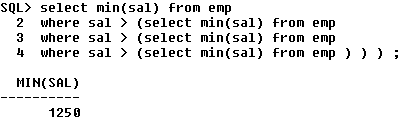
**6) Display the 2nd maximum salary**

****

**7) Display the 3rd maximum salary**

****

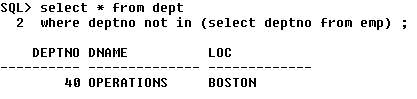
**8) Display the 4th least salary**

****

This method is not efficient to find the maximum and minimum salary. The limit is 32. This is not efficient if you want to find the 100th maximum salary.

**We can have upto 32 levels of sub-queries only.**

**9) List the department names that are having no employees at all**



***CHAPTER 7***

**JOIN**

**Joins** are used when we need to fetch the data from multiple tables

**Types of JOIN(s)**

* Cartesian Join (product)
* Inner (Equi) Join
* Outer Join - Left Outer Join, Right Outer Join, Full Outer Join
* Self Join

**CARTESIAN JOIN**

- It is based on Cartesian product theory.

**Cartesian Product Theory** in Mathematics states that :-

Let there be two sets – A {1, 2, 3} & B {4, 5}

Thus the Cartesian product (A\*B) will be,

A \* B = { (1,4), (1,5), (2,4), (2,5), (3,4), (3,5) }

Thus there are 6 sets – order of A is 3 & order of B is 2. Therefore, 2\*3 = 6 is the Cartesian product.

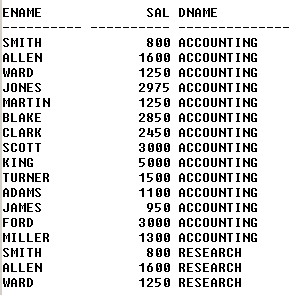
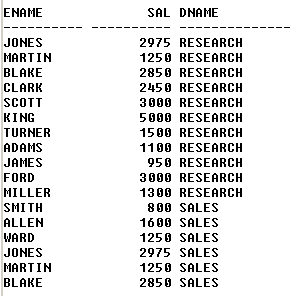
Here, each and every record of the 1st table will combine with each and every record of the 2nd table.

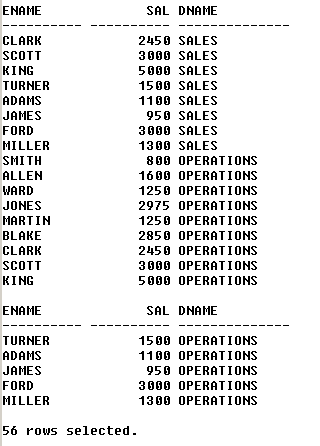
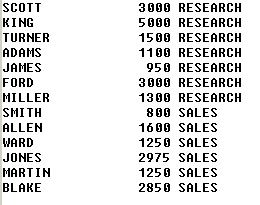
If a table A is having 10 records & B is having 4 records – the Cartesian join will return 10\*4 = 40 records.

**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**.

**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.

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

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

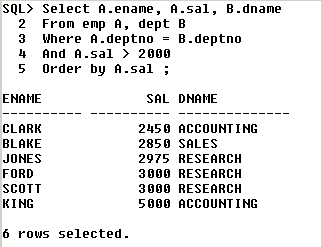
From emp A, dept B

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 ;

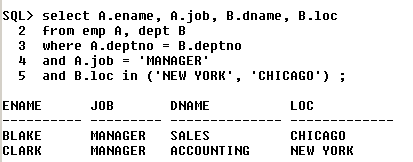
**Therefore, Number of JOINS = Number of common columns**

If there are no common columns, then reject it saying that the two tables can be joined.

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

**For ex –** customerid & cid

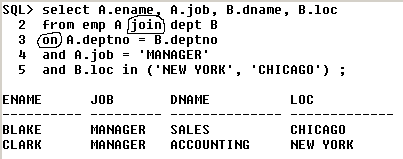
**Display employee name, his job, his dname and his location for all the managers living in New York or Chicago**



**ANSI style JOINS**

This was introduced from Oracle 9i.

It is another way of writing inner joins with a few modifications.

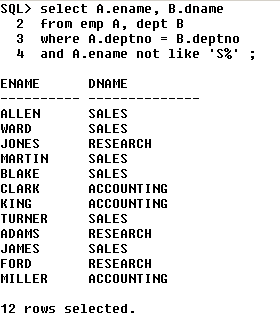


Thus we, can see the changes ,

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

**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

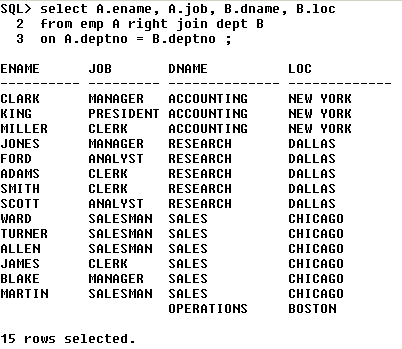
Outer join = inner join + non-matching records

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.

**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***

****

**A B**

**10 6**

**3 3**

**7**

**3**

**A CJ B = 60records A IJ B = 3records(3 matching)**

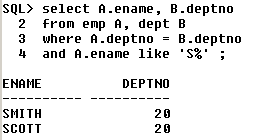
**A LJ B = 10records (3matching + 7non matching of A)**

**A RJ B = 6records (3matching + 3non matching of B)**

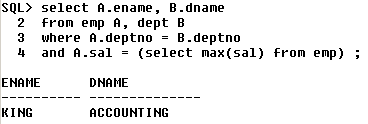
**A FJ B = 13records (3matching of A & B + 7nonmatching of A + 3nonmatching of B)**

**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**

Joining a table to itself is called self join

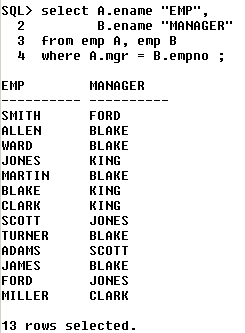
The **FROM** clause looks like this,

FROM emp A, emp B

Or

FROM emp A join emp B - *ANSI style*

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

****

**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 |

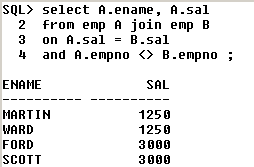
|  |  |  |
| --- | --- | --- |
| **Emp (B)** | | |
| **EmpNo** | **Ename** | **Mgr** |
| 101 | Scott | 102 |
| 102 | Blake | 103 |
| 103 | King | - |
| 104 | Smith | 103 |
| 105 | Jones | 104 |

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**



**Co – related Queries :**

* They are special type of sub – queries
* Here, both outer & inner queries are inter-dependent
* For each & every record of outer query, the entire inner query will be executed
* They work on the principles of both **sub – queries & JOIN(s)**.

**For ex, Display the employee who is earning the highest salary**

****

****

Thus, if an outer query column is being accessed inside the inner query, then that query is said to be co-related.

Let us see the logic i.e, how we get the 1st max salary :-

|  |  |  |
| --- | --- | --- |
| **Emp (A)** | | |
| **EmpNo** | **Ename** | **Sal** |
| 101 | Scott | 3000 |
| 102 | Blake | 4000 |
| 103 | King | 5000 |
| 104 | Smith | 2000 |
| 105 | Jones | 1000 |

|  |  |  |
| --- | --- | --- |
| **Emp (B)** | | |
| **EmpNo** | **Ename** | **Sal** |
| 101 | Scott | 3000 |
| 102 | Blake | 4000 |
| 103 | King | 5000 |
| 104 | Smith | 2000 |
| 105 | Jones | 1000 |

Since co-related queries are a combination of Joins and sub-queries.

It follows the concept of Joins and creates multiple copies of the same table.

Then it takes 1st record i.e, - Blake – sal is 3000. It starts comparing with the sal in the emp table,

3000 = 3000 - count starts from 0 – thus, 0 = 0

3000 < 4000 – thus, 0 ! = 1

3000 < 5000 – thus, 0 ! = 2

3000 > 2000 – thus , 0! = 2

3000 > 1000 – thus, 0 ! = 2 if the condition becomes false, then the count increments by 1. Here 3000 is less than 4000 & 5000, thus 0 ! = 2. Thus , Blake does not have the highest salary.

Similarly, it does for the next records,

Blake – salary of 4000 – but 4000 < 5000 – thus, 0 ! = 1. This is also false.

King – salary of 5000 – it is greater than everything – thus, 0 = 0. Thus, King has the highest salary.

But the query doesn’t stop here, it checks for Smith & Jones as well.

Similarly, if we want to find the 2nd maximum salary,

Then in the query, change ‘0’ to ‘1’ & here, the logic is – it compares until it gets 1 = 1.

For 3rd maximum salary – change 0 to 2 and so on – here, the logic is – it compares until it gets 2 = 2.

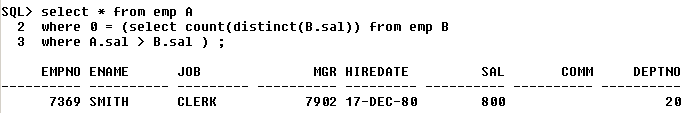
**For any highest, always put it as ‘0’ in the query.**

**If you want n(th) salary, pass (n-1).**

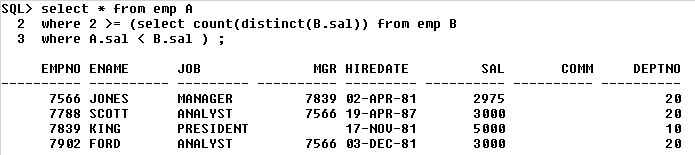
**In interview – this is a definite question. They will ask you what is co-related queries. And then they’ll ask you find, 1st or max or 3rd maximum salary – after you write the query – they will ask you to explain the logic as to how it gets the same – draw the table and explain it to them just as shown above.**

**Assignment**

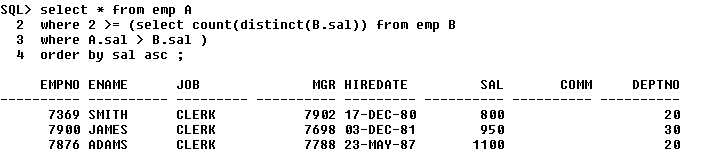
**1) Display the least salary from the employee table.**



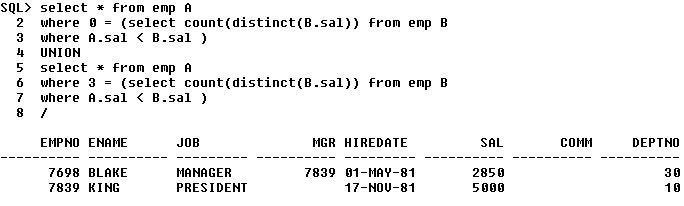
**2) Display top 3 person’s salaries from the employee table.**



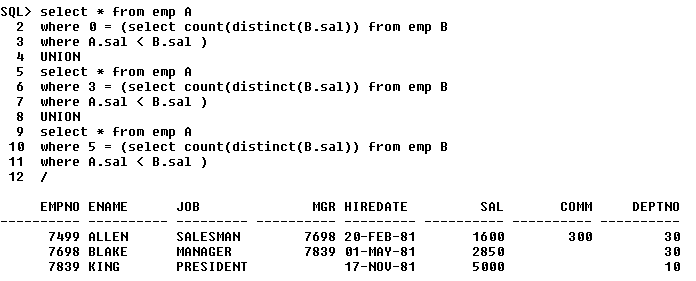
**3) Write a query to display bottom 3 salaries**

****

**4) Display 1st and 4th maximum salary**

****

**5) Display 1st, 4th & 6th highest salaries in a single query**

****

***CHAPTER 8***

**FUNCTIONS**

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

There are **2 types**,

* Pre – defined
* User defined

**Pre – defined**

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

These are used both in SQL and PL/SQL. PL – Procedural Language (it’s a extension to SQL, can contain IF statements, loops, exceptions, OOPs, etc .. )

**User – defined**

Used only in PL/SQL and we will not study it here.

We have already learnt about GROUP functions.

Now, let us study the various CHARACTER functions.

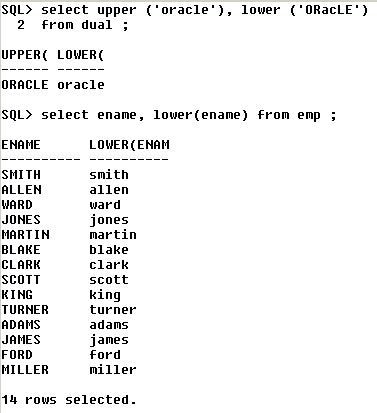
**CHARACTER functions**

a) Upper

b) Lower

c) Length

**For ex :-**

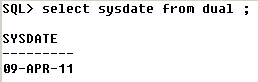


In the 1st query, we see something called as **dual**.

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

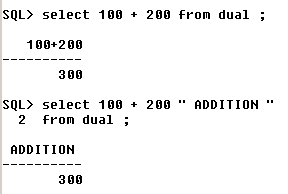
**For ex,**

1)

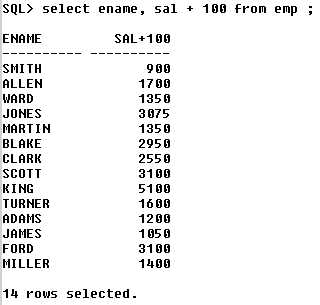


This gives the system date.

2)



3)

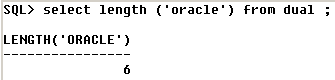


We use dual – when the data is not present in any of the existing tables. Then we use dual.

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

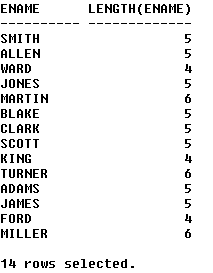
**For ex,**

**1)**

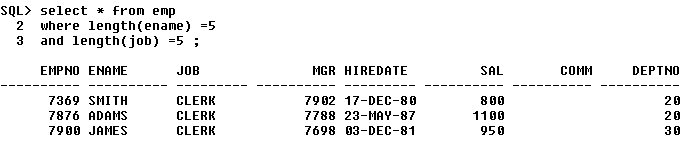


**2)**





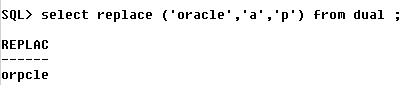
**3) Display all the employees whose name & job is having exactly 5 characters**



**REPLACE**

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

**For ex,**

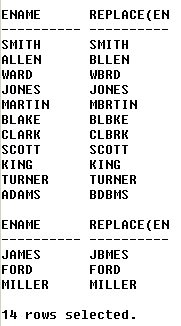


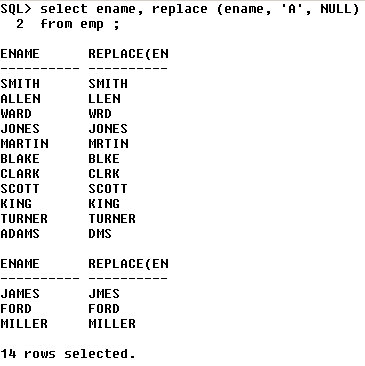
Here, **a –** is the old value to be replaced with **p** – which is the new value.



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

Let us see the output as shown below,



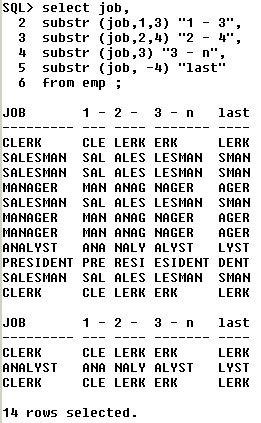


**SUBSTR**

This is called **substring**.

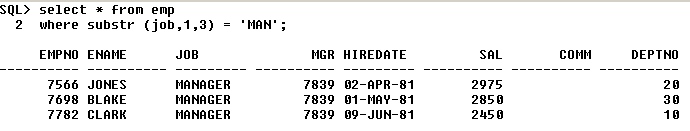
It extracts ‘n’ characters from x(th) position of a given string.

**For ex,**



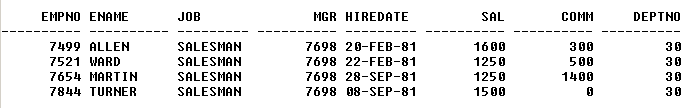
Here **, (job, ‘1’ , ‘3’)** – means from **job** – extract **from 1st position , 3 characters**.

**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,**

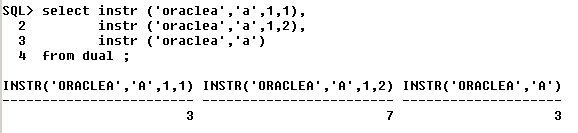
***Select instr (‘oracle’ , ‘a’ , 1 , 1) from dual ;***

Given string

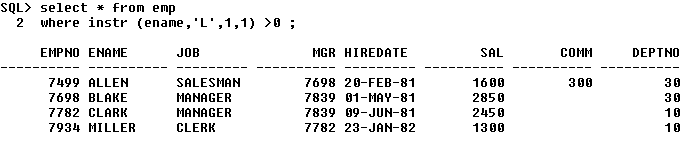
Character to be searched

Position from where the search should begin

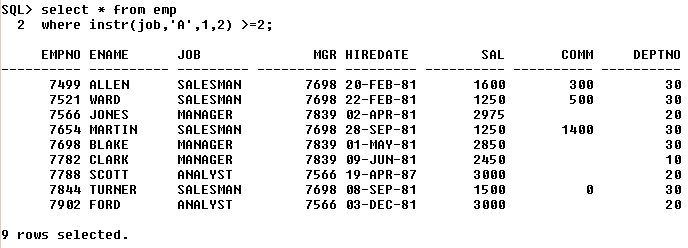
Number of occurences



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



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

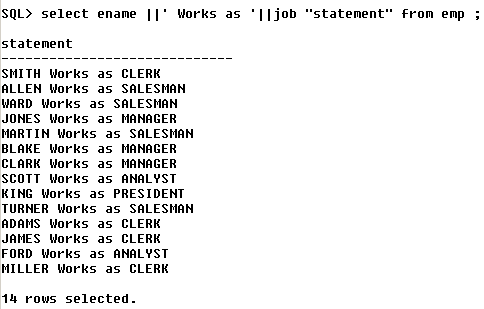


**CONCAT**

It concatenates any two values or columns.

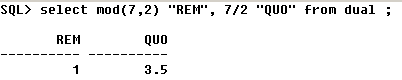
It is represented by - **||**

**For ex,**

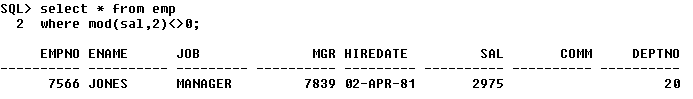


**NUMERIC FUNCTIONS**

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



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

****

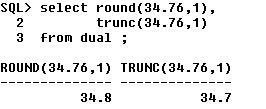
**Round**

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**

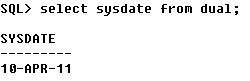
**1) Sysdate**

Stands for System date.

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

The default format is,

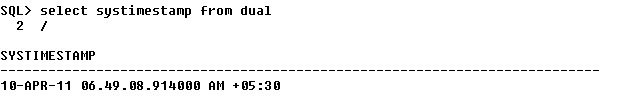
**dd – mon – yy**

****

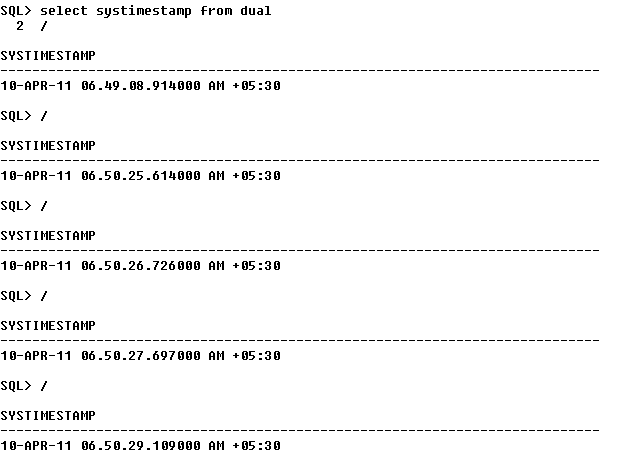
**2) Systimestamp**

Introduced from Oracle 9i

Returns date, time and timezone.

****

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

****

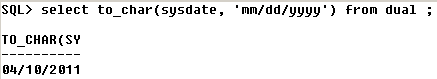
**In interview – if they ask you – *“ which function contains fractions of a second “ OR “how to see the system time “* – then answer is “SYSTIMESTAMP”.**

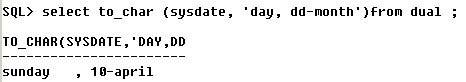
**SPECIAL FUNCTIONS**

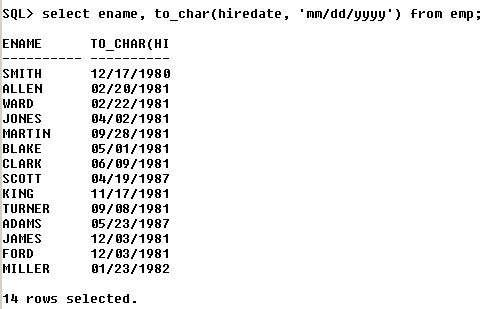
**1) TO – CHAR**

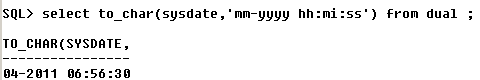
Used for displaying the date in different formats.

**For ex,**

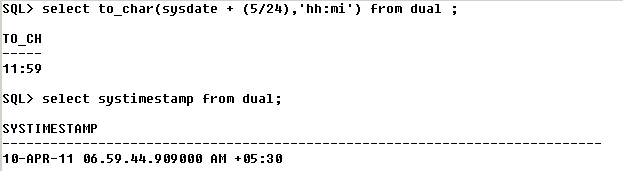
****

****

****

****

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

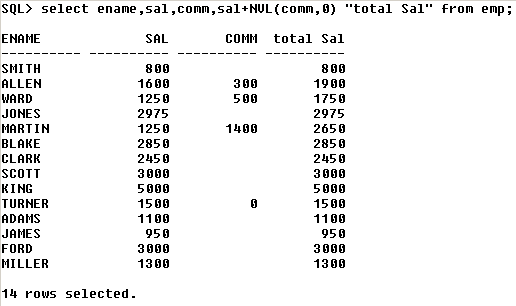
****

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

**NVL**

It substitutes a value for a null.

**For ex,**

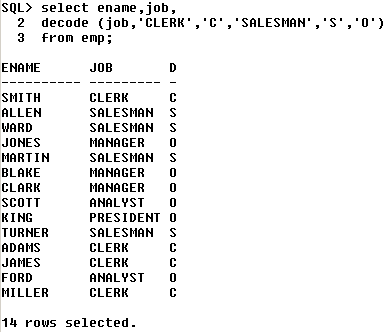
****

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.

**DECODE**

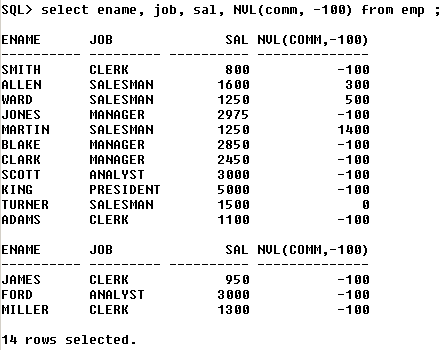
It works like ‘**if – then – else**’ statement.

**For ex,**

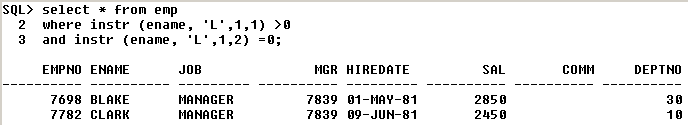


The above query states that – in job, if clerk is there, replace with C – else if salesman is there, replace it with S – else replace with ‘O’.

**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**

****

***CHAPTER 9***

**NORMALIZATION**

**Normalization** is the process of splitting the bigger table into many small tables without changing its functionality.

It is generally carried out during the design phase of SDLC.

**Advantages**

1) it reduces the redundancy (unnecessary repeatation of data)

2) avoids problem due to delete anamoly (inconsistency)

Normalization is a step-by-step process and in each step, we have to perform some activities.

**STEPS IN NORMALIZATION**

1) 1NF – 1st Normal form

2) 2NF – 2nd Normal form

3) 3NF – 3rd Normal form

**1NF**

- We should collect all the required attributes into 1 or more bigger entities.

- We have to assume no 2 records are same (i.e, records should not be duplicated)

- Identify the probable primary key

At the end of 1NF, our data looks like this,

|  |  |
| --- | --- |
| **COLLEGE** | |
|
| RegNo **- PK** | |
| Sname |  |
| Semester |  |
| DOB |  |
| MailID |  |
| Phone |  |
| BookNo - **PK** | |
| Bname |  |
| Author |  |
| DOI |  |
| DOR |  |
| Fine |  |

**2NF**

To perform 2NF,

- The tables have to be in 1NF

- Here, we identify all the complete dependencies and move them separately into different tables.

At the end of 2NF, our data looks like this,

|  |  |  |  |
| --- | --- | --- | --- |
| **STUDENTS** | |  | **BOOKS** |
|  |
| RegNo - PK | |  | BookNo - PK |
| Sname | |  | RegNo - FK |
| Semester | |  | Bname |
| DOB | |  | Author |
| MailID | |  | DOI |
| Phone | |  | DOR |
|  |  |  | Fine |

**3NF**

The table will have to be in 2NF

Here, we identify all the partial dependencies and move such columns to a separate table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **STUDENTS** |  | **BOOKS** | |  | **LIBRARY** | |
|  |  |
| RegNo - PK |  | BookNo - PK | |  | BookNo - FK | |
| Sname |  | Bname | |  | RegNo - FK | |
| Semester |  | Author | |  | DOI | |
| DOB |  |  |  |  | DOR | |
| MailID |  |  |  |  | Fine | |
| Phone |  |  |  |  |  |  |

**Disadvantage of Normalization**

The only minor disadvantage is we may have to write complex queries as we have more number of tables to be accessed.

**Denormalization** is the process of combining more than 1 smaller table to form 1 bigger table is called as denormalization.

**CODD rules** ( **Differentiates between DBMS & RDBMS** )

1) should support NULL values

2) should support creation of relationship between tables

3) should support DDL, DML, TCL

4) should support constraints like PK, Unique, CHK

5) should support query techniques like sub – queries, joins, grouping etc.

**Oracle 9i Features(i means internet)**

- TIMESTAMP datatype

- SYSTIMESTAMP function

- ANSI style joins

- Renaming a column

**Oracle 10g features (g means grid)**

- Recycle Bin

**ERD - Entity Relationship Diagram**

It is the pictorial representation of all the entities and their relationships (tables).

|  |
| --- |
| **STUDENTS** |
| RegNo - PK  Sname  Semester  DOB  MailID  Phone |

|  |
| --- |
| **STUDENTS \_ INTERNALS** |
| RegNo - FK  Sid  Marks |

|  |
| --- |
| **BOOKS** |
| BookNo - PK  BName  Author |

|  |
| --- |
| **LIBRARY** |
| BookNo - FK  RegNo - FK  DOI  DOR  Fine |

**VERY IMPORTANT**

**Phone No. of Chethan Sir – 9741120618**

**Email id of chethan sir – chethanfig @ yahoo.com**

**\*\*\*\*\* THE END \*\*\*\*\***