SQL

Database V/S DBMS

- **Database:**
  - It is collection of meaningful data.
  - It is group of objects(table, views, synonyme,UDF, Procedure, trigger etc...)
- **DBMS:**
  - DBMS is management system which manages the data base objects and data as well.
- **Activities**
  - 1. Create
  - 2. Inserting data
  - 3. Deleting data from the objects
  - 4. Retrieve/select data from the objects

Database software

- The software is used for doing DBMS/RDBMS activities is called database.
- RDBMS(Relational Database Management System)
- Relationships between the database Objects.
- DBMS
  - Ex: MS-Access, D-Base, FoxPro etc...
- RDBMS:-
  - Ex: Oracle, MS-SQL Server, IBM DB2, MySQL (free source), Sybase, Teradata etc....

Database Components

- 1. DB Client
- 2. DB Server
- DB Client is a interface used for connecting to database(CLI or GUI).
- DB Server is a storage area where the database objects and data will be stored.
- SQL is a Language used to communicate with the database objects.

Oracle Download/Connection

- Downloading Oracle
- http://www.oracle.com/technetwork/indexes/downloads/index.html#database
- Connecting to the database
  - Host Name: INVRLX61ILM40
  - Port : 1521
  - SID/Service ID: ORA11G
  - Username:hr
  - Password: hr

SQL Language

1. DDL(Data Definition Language)
2. DML(Data Manipulation Language)
3. DRL/DQL (Data Retrieval Language/Data Query Language)
4. TCL (Transaction Control Language)
5. DCL (Data Control Language)

SQL Commands

- DDL:
  - create, alter, drop, truncate, rename
- DML:
  - insert, update, delete
- DRL/DQL:
  - select
- TCL
  - Commit, Rollback, save point
- DCL
  - GRANT, REVOKE

Creating Table

- create table <<TABLE NAME>>(col1 datatype,col2 datatype, col3 datatype.....);

- Ex:
- **CREATE TABLE STUDENT(SNO NUMBER(5),SNAME VARCHAR(15),MARKS NUMBER(3));**

Inserting data into table

- INSERT INTO <<TABLE NAME>> VALUES(VAL1,AL2,VAL3....);
- **INSERT INTO STUDENT VALUES(101,'kiran',80);**
- **INSERT INTO STUDENT(SNAME,SNO,MARKS) VALUES('RAM',102,60);**
- **INSERT INTO STUDENT VALUES(103,'kRISHNA',NULL);**
- **INSERT INTO STUDENT VALUES(&SNO,&SNAME,&MARKS);**

Selecting Rows from a table

- **SELECT * FROM EMPLOYEES;**
- **SELECT EMPLOYEE_ID,FIRST_NAME,SALARY FROM EMPLOYEES;**
- **SELECT EMPLOYEE_ID EMPID,FIRST_NAME FNAME,SALARY+300 SAL FROM EMPLOYEES;**

SQL Data types

- **CHAR/ VARCHAR**
  - ename char(5)----->'pavan;
  - ename char(5)---> 'pa'
  - ename varchar2(5)----'pavan'
  - ename varchar2(5)----'pa'
- **NUMBER**
  - salary number(5)
  - length number(5,3) // scale & precision
- **DATE**
  - HireDate Date

SQL Data types…

- **LONG**
  - Similar to varchar2 type but allows 2 GB
- **RAW**
  - Used to store images, voice, videos files, text files etc.....
- **LONGRAW**
  - similar to RAW datatypes
- **CLOB, BLOB, BFILE** etc.....

Where clause

- Used for selecting the rows based on condition.(Filtering the rows using where condition)
- **SELECT * FROM EMPLOYEES;**
- **SELECT * FROM EMPLOYEES WHERE SALARY>3000;**
- **SELECT * FROM EMPLOYEES WHERE SALARY<=3000;**
- **SELECT * FROM EMPLOYEES WHERE DEPARTMENT_ID=30;**
- **SELECT * FROM EMPLOYEES WHERE COMMISSION_PCT is null;**
- **SELECT * FROM EMPLOYEES WHERE FIRST_NAME='Jennifer';**
- **SELECT DISTINCT DEPARTMENT_ID FROM EMPLOYEES;**
- **SELECT distinct * FROM EMPLOYEES;**

Updating data into table

- **UPDATE STUDENT SET MARKS=50 WHERE MARKS IS NULL;**
- **UPDATE STUDENT SET SNAME='PAVAN',MARKS=70 WHERE SNO=106;**

Logical Operators

- **AND**
- **OR**
- **NOT**
- **SELECT * FROM EMP WHERE SAL>1000 AND JOB='CLERK';**

- **SELECT * FROM EMP WHERE SAL>2000 OR JOB='CLERK';**

- **SELECT * FROM EMP WHERE NOT ENAME='SMITH';**

Between & IN Operators

- **Between** --> used to display the rows which is following in the range of values.
- **Not Between**
- **SELECT * FROM EMP WHERE SAL BETWEEN 2000 AND 5000;**
- **SELECT * FROM EMP WHERE SAL NOT BETWEEN 2000 AND 5000;**

- **IN** --> IN operator return the rows when the values are matching in the list
- **Not In**
- **SELECT * FROM EMPLOYEES WHERE SALARY=3600 OR SALARY=4000 OR SALARY=3900;**
- **SELECT * FROM EMP WHERE SAL IN(800,1600,1300);**
- **SELECT * FROM EMP WHERE SAL NOT IN(800,1600,1300);**

PATTERN MATCHING OPEARATORS(Whiled card characters)

- **%** --> many characters
- **_** --> single character

- **SELECT * FROM EMPLOYEES WHERE FIRST_NAME LIKE 'S%';**
- **SELECT * FROM EMPLOYEES WHERE FIRST_NAME LIKE '%r';**
- **SELECT * FROM EMPLOYEES WHERE FIRST_NAME LIKE 'S%r';**
- **SELECT * FROM EMPLOYEES WHERE FIRST_NAME LIKE '%m%'**
- **SELECT * FROM EMPLOYEES WHERE FIRST_NAME NOT LIKE 'S%';**
- **SELECT * FROM EMPLOYEES WHERE FIRST_NAME LIKE '%e_';**
- **SELECT * FROM EMPLOYEES WHERE FIRST_NAME LIKE '___';**

DDL Commands(Data Definition Language)

1) CREATE
2) ALTER
3) DROP
4) TRUNCATE
5) RENAME

Create & alter

- CREATE is used to create database objects(Table, views, synonymes etc...)
- ALTER

1. Adding a new column
2. Dropping the existing column
3. Modifying the existing column (Increase/Decrease size of the column & change the data type of the column)
4. Renaming a column

- **Adding a new column**
- ALTER TABLE STUDENT ADD(grade varchar(2));
- **Dropping a column from table**
- ALTER TABLE STUDENT DROP(GRADE);
- **Modifying the existing column**
  - We can increase/decrease the size of the column.
  - We can decrease the column size ONLY when existing column values can fit into new size..
  - Column should be empty should be empty to change its data type.
- ALTER TABLE STUDENT MODIFY(GRADE NUMBER(2));
- **Renaming a column**
- ALTER TABLE STUDENT RENAME COLUMN SNAME TO STUNAME;

- DROP:
- Used to dropping the table definition with data
- **DROP TABLE STUDENT;**
- TRUNCATE
- Used to remove all the rows from the table.
- **TRUNCATE TABLE STUDENT;**
- DELETE
- Used for deleting all the rows from the table.
- **DELETE FROM TABLE;**

Differences between Drop, Truncate & delete

- **DROP TABLE STUDENT;** -- Drops the structure & data
- **TRUNCATE TABLE STUDENT;** -- Removes the all the rows permanently
- **DELETE FROM STUDENT;** -- Removes the all the rows temporarily. We can Roll back the rows.

- **RENAME**
- used for changing the name of the table
- **RENAME STUDENT TO STU;**

SQL Functions

- 1. Built-in Functions
- 2. User Defined Functions (PL/SQL)

- There are 2 types of Built-in functions
  1. Group Functions (Multiple row functions)
  2. Scalar Functions (Single row functions)

- **Dual Table:** it is a Dummy table generally used for some calculations. It has one row and one column.

Group Functions

- **AVG()**
- **SUM()**
- **MAX()**
- **MIN()**
- **MAX()**
- **COUNT()**
- **SELECT AVG(SALARY) FROM EMPLOYEES;**
- **SELECT SUM(SALARY) FROM EMPLOYEES;**
- **SELECT MIN(SALARY) FROM EMPLOYEES;**
- **SELECT MAX(SALARY) FROM EMPLOYEES;**
- **SELECT COUNT(*) FROM EMPLOYEES;**
- **sysdate**: provides current system date.
- **select sysdate from dual;**

Scalar Functions

- 1. Character functions
- 2. Number/Numeric Functions
- 3. Date Functions
- 4. Conversion Functions

Character Functions

- Upper(): converts into upper case letters.
- Lower() : converts into lower case letters.
- Initcap(): converts first letter is capital and remaining are lowqer case letters.

- **SELECT UPPER(First_name) from employees;**
- **SELECT LOWER(First_name) from employees;**
- **SELECT INITCAP(First_Name) from employees;**

- Length(): return the length of string.

- **SELECT LENGTH('oracle') from dual;**
- **SELECT * FROM EMPLOYEES WHERE LENGTH(FIRST_NAME)=4;**

- LDAP(): Pads the character towards the left side.
- RPAD(); Pads the character towards the right side.

- **SELECT RPAD('ORACLE',10,'XXX') FROM DUAL;** // ORACLEZZZZ
- **SELECT LPAD('ORACLE',10,'YYY') FROM DUAL;** //YYYYORACLEF

- TRIM(): Removes the specified characters from both sides.
- **SELECT TRIM(' ORACLE ') FROM DUAL;**
- **SELECT TRIM('z' from 'zzoraclezz') from dual;**

- INSTR(): Returns the position of the character within a string.
- **SELECT INSTR('ORACLE','E') FROM DUAL;**

- SUBSTR(): Returns the substring of the string.
- **SELECT SUBSTR('ORACLE',2,3) FROM DUAL;** //RAC
- **SELECT SUBSTR('ORACLE',3,3) FROM DUAL;** //ACL
- **SELECT SUBSTR('ORACLE',4,3)FROM DUAL;** //CLE
- **SELECT SUBSTR(FIRST_NAME,1,3)||'****' FROM EMPLOYEES;**

- CONCAT(): To join two strings.

- **SELECT CONCAT('ORACLE','TRAINING') FROM DUAL;**
- **SELECT CONCAT(FIRST_NAME,LAST_NAME) ENAME FROM EMPLOYEES;**

Number/Numeric Functions

- abs(): return absolute value.
- **SELECT ABS(-40) FROM DUAL;**
- **SELECT ABS(40) FROM DUAL;**
- sqrt(): returns square root of provided value.
- **select SQRT(25) from dual;**
- Mod(): return reminder value
- **select MOD(10,3) FROM DUAL;** //1
- Power(): return power value (2*2*2*2*2)
- **select power(2,5) from dual;**

- Trunc(): removes the decimal points.

- **select TRUNC(40.9) FROM DUAL;** // 40
- **select TRUNC(40.1234,3) FROM DUAL;** // 40.123
- **select TRUNC(40.1234,2) FROM DUAL;** // 40.12
- **Select TRUNC(6876,-1) FROM DUAL;** //6870
- **Select TRUNC(6876,-2) FROM DUAL;** //6800
- **Select TRUNC(68763456,-5) FROM DUAL;** //68700000

- Greatest() & Least(): returns greatest, least values in the provided values.

- **select GREATEST(100,200,300,400,500) FROM DUAL;**
- **SELECT LEAST(100,200,300) FROM DUAL;**

Date Functions

- ADD_MONTHS()
- MONTHS_BETWEEN()
- NEXT_DAY()
- LAST_DAY()

- ADD_MONTHS(): this will add months to provided date.

- **SELECT ADD_MONTHS(sysdate,6) from dual;**
- **SELECT ADD_MONTHS('15-FEB-2016',8) from dual;**

- Months_between(): returns number of months between given dates.

- **SELECT MONTHS_BETWEEN(SYSDATE,'10-JULY-2015') FROM DUAL;**

- **SELECT FIRST_NAME,TRUNC(MONTHS_BETWEEN(SYSDATE,HIRE_DATE))“ Exp in Months" FROM EMPLOYEES;**

- **SELECT FIRST_NAME,trunc(MONTHS_BETWEEN(SYSDATE,HIRE_DATE)/12) “ Exp in Years" FROM EMPLOYEES;**

- Next_Day(): returns date of the specified date
- **SELECT NEXT_DAY(sysdate,'friday') from dual;**
- Last_day(): Returns the last day of the month.
- **select last_day('01-Feb-2016') from dual;**

Conversion Functions

- TO_CHAR()
- TO_NUMBER()
- TO_DATE()

- To_char():
- **SELECT FIRST_NAME,EMPLOYEE_ID,HIRE_DATE FROM EMPLOYEES;**
- **SELECT FIRST_NAME,EMPLOYEE_ID, TO_CHAR(HIRE_DATE,'DD-MM-YYYY') from employees;**
- **SELECT FIRST_NAME,EMPLOYEE_ID, TO_CHAR(HIRE_DATE,'DD-MON-YYYY') from employees;**
- **SELECT SYSDATE FROM DUAL;**
- **SELECT TO_CHAR(SYSDATE,'DAY') FROM DUAL;**
- **SELECT TO_CHAR(SYSDATE,'DD') FROM DUAL;**
- **SELECT TO_CHAR(SYSDATE,'MON') FROM DUAL;**
- **SELECT TO_CHAR(SYSDATE,'MM') FROM DUAL;**
- **SELECT TO_CHAR(SYSDATE,'YYYY') FROM DUAL;**
- **SELECT TO_CHAR(SYSDATE,'YY') FROM DUAL;**

- // Display employees who are joined in 1998.

- **select * from employees where to_char(Hire_Date,'yy')='98';** //Extract year
- **select * from employees where to_char(Hire_Date,'yyyy')='1998';**

- // Dispaly employees who are joined in FEB

- **select to_char(hire_date,'MON') from employees;** // Extracts Month
- **select * from employees where to_char(hire_date,'MON')='FEB';**

- TO_NUMBER()
- **SELECT LTRIM('$1400','$') FROM DUAL;** //1400 is in string format
- **SELECT TO_NUMBER(LTRIM('$1400','$')) FROM DUAL;** //1400 is in Number format

- To_Date()
- **select TO_DATE('09-JAN-16') FROM DUAL;**
- **select TO_DATE('09/JANUARY/2016') FROM DUAL;**

Group By clause

- **group By clause**: is used to divide rows into several groups.
- select department_id,sum(salary) from employees group by department_id;
- select department_id,avg(salary) from employees group by department_id ;
- select department_id, max(salary),min(salary) from employees group By department_id;
- select Job_id, count(*) from employees group by job_id;

- ***All the columns in the select list should include in group by clause.***

- **select department_id, job_id, sum(salary) from employees group by department_id, job_id;**
- **select department_id,sum(salary),first_name from employees group by department_id; // invalid query**
- **Duplicate rows:**
- **Select ENO, ENAME, SAL, COUNT (*) FROM EMP GROUP BY ENO, ENAME, SAL HAVING COUNT (*) >1;**

Having & Order by clause

- Having clause: Having clause is used to filter the output from the group by clause.
- **select department_id, sum(salary) from employees where department_id<>50 group by department_id;**
- **select department_id, sum(salary) from employees group by department_id having sum(salary)>20000;**

- Order By clause: Order by clause is used to arrange the rows in a table (ascending or descending order).
- **select * from employees order by department_id desc;**
- **select * from employees order by salary;**

Order of execution

- Order of execution
- **Where-> group by-> having-> order by**

- select department_id,sum(salary) from employees group by department_id having sum(salary)>20000 order by sum(salary);

- select department_id,sum(salary) from employees where department_id<>100 group by department_id having sum(salary)>20000 order by sum(salary)desc ;

Integrity Constraints

- Constraints are rules can apply on columns.

- **1) NOT NULL**
- **2) UNIQUE**
- **3) PRIMARY KEY**
- **4) FOREIGN KEY or REFERENCECIAL INTEGRITY**
- **5) CHECK**

Not null

- NOT NULL: This is a constraint will not accept NULL values into the column.

- You can apply NOT NULL on any number of columns

```
create table student1( sno number(3) NOT NULL,
                       sname varchar(10),
                        marks number(3));
```

- **insert into student1 values(101,'arun',50);** // CORRECT
- **insert into student1 values(NULL,'KIRAN',70);** // ERROR

UNIQUE

- UNIQUE: this constraint will not accept duplicate values.
- This constraint can apply on both **column and table level**.

- //column level

```
create table student1( sno number(3) Unique,
                       sname varchar(10),
                       marks number(3));
```

- //Table level

```
create table student1( sno number(3,
                       sname varchar(10),
                       marks number(3),
                       unique(sno)
                        );
```

- **insert into student1 values(101,'arun',50);**
- **insert into student1 values(101,'kiram',60);** // 101 not allowed

- **insert into student1 values(null,'suresh',80);**
- **insert into student1 values(null,'raj',60);**

- * **Unique constraint column can accept multiple NULLS.**

Primary Key

- PRIMARY KEY : Combination of Unique + Not Null
- **primary key column will not allow duplicate values and also null values.**
- **primary key constraint can create both column level & table level.**

```
create table student1(sno number(3) primary key,
                      sname varchar(2),
                      marks number(3));
```

- Alter table student1 modify(sname varchar(10));
- insert into student1 values(101,'arun',50); // right
- insert into student1 values(101,'suresh',60); // Invalid
- insert into student1 values(null,'suresh',60); // Invalid

- **we can create primary key on combination of two columns called as composite primary key.**
- **Composite key acan be applied only at table level.**

FOREIGN KEY CONTRAINT (OR) REFERENCIAL INTEGRITY

- // parent Table

```
create table school(sno number(3),
                    sname varchar(15),
                    marks number(3),
                    primary key(sno));
```

- insert into school values(101,'arun',90);
- insert into school values(102,'kiran',70);
- insert into school values(103,'amit',80);
- Select * from school;

- // child

```
create table library(sno number(3) references school(sno),
                     book_name varchar2(10));
```

- insert into library values(102,'java'); // valid
- insert into library values(108,'c'); // in valid
- insert into library values(null,'dot net'); //valid

ON DELETE CASCADE

- ON DELETE CASCADE: We can delete the rows from the parent table and the corresponding child table rows deleted automatically.

```
//Parent Table
CREATE TABLE SCHOOL( SNO NUMBER(3),
                     SNAME VARCHAR2(15),
                     MARKS NUMBER(3),
                     PRIMARY KEY(SNO));
)
```

- **INSERT INTO SCHOOL VALUES(101,'ARUN',50);**
- **INSERT INTO SCHOOL VALUES(102,'AJAY',60);**
- **INSERT INTO SCHOOL VALUES(103,'KIRAN',80);**

```
CREATE TABLE LIBRARY( SNO NUMBER(3)REFERENCES SCHOOL(SNO) ON DELETE CASCADE ,
                      BOOK_NAME VARCHAR2(10));
```

- **INSERT INTO LIBRARY VALUES(101,'DOT NET');**
- **INSERT INTO LIBRARY VALUES(102,'JAVA');**
-

- **SELECT * FROM SCHOOL;**
- **DELETE FROM SCHOOL WHERE SNO=102; //valid**
- **SELECT * FROM LIBRARY;**

- One row deleted from parent table and one from child table also deleted.

Foreign key constraint at table level

```
CREATE TABLE SCHOOL( SNO NUMBER(3),
                SNAME VARCHAR2(15),
                MARKS NUMBER(3),
                PRIMARY KEY(SNO));
)
```

- **INSERT INTO SCHOOL VALUES(101,'ARUN',50);**
- **INSERT INTO SCHOOL VALUES(102,'AJAY',60);**
- **INSERT INTO SCHOOL VALUES(103,'KIRAN',80);**

```
CREATE TABLE LIBRARY( ROLLNO NUMBER(3),
                BOOK_NAME VARCHAR2(10) FOREIGN KEY (ROLLNO) REFERENCES SCHOOL(SNO) ON DELETE CANCADE;
```

- **INSERT INTO LIBRARY VALUES(101,'DOT NET');**
- **INSERT INTO LIBRARY VALUES(102,'JAVA');**

Check Constraint

- Check constraint is used for allowing to user to enter specific values into column.

```
create table student(
sno number(5),
sname varchar2(15),
marks number(5) check(marks between 50 and 100));
```

- insert into student values(101,'amith',60); // valid
- insert into student values(101,'amith',45);// in valid
- insert into student values(101,'amith',105); //invalid

```
create table loc
( city varchar(15) check(city in('HYDERABD','CHENNAI','DELHI')),
  country varchar(15),
  pin number(8));
```

- insert into loc values('HYDERABD','INDIA',123456); // VALID
- insert into loc values('MUMBAI','INDIA',644566); // in valid
- insert into loc values('DELHI','INDIA',678445); //valid
- select * from loc;

Data Dictionary & Metadata

- There are 2 types of tables:

1. system defined tables (TAB,USER_CONSTRAINTS)

2. user defined tables (EMP, SCHOOL ETC...)

- USER_CONSTRAINTS --> is a system table contains metadata.
- **Meta data:** data about the data of database objects.
- **Data dictionary :** The columns present on system tables is called data dictionary.
- **SELECT * FROM TAB; // Displays all database objects present in your current database.**
- **SELECT TABLE_NAME,CONSTRAINT_NAME,CONSTRAINT_TYPE FROM USER_CONSTRAINTS WHERE TABLE_NAME='SCHOOL';**

Adding & Dropping Constraints

- **Adding constraints to existing table:**
- ALTER TABLE STUDENT1 ADD(PRIMARY KEY(SNO)); // ADDING CONSTRAINT FOR EXISTING COLUMN IN A TABLE

- **Dropping constraint**
- ALTER TABLE STUDENT1 DROP PRIMARY KEY;

SET OPERATORS

- SET Operators are used for getting data from multiple tables.
- Columns should be the same data type in all the tables.
- **UNION**
- **UNION ALL**
- **INTERSECT**
- **MINUS**

create table A
(SNAME VARCHAR2(10), NUM NUMBER(2));

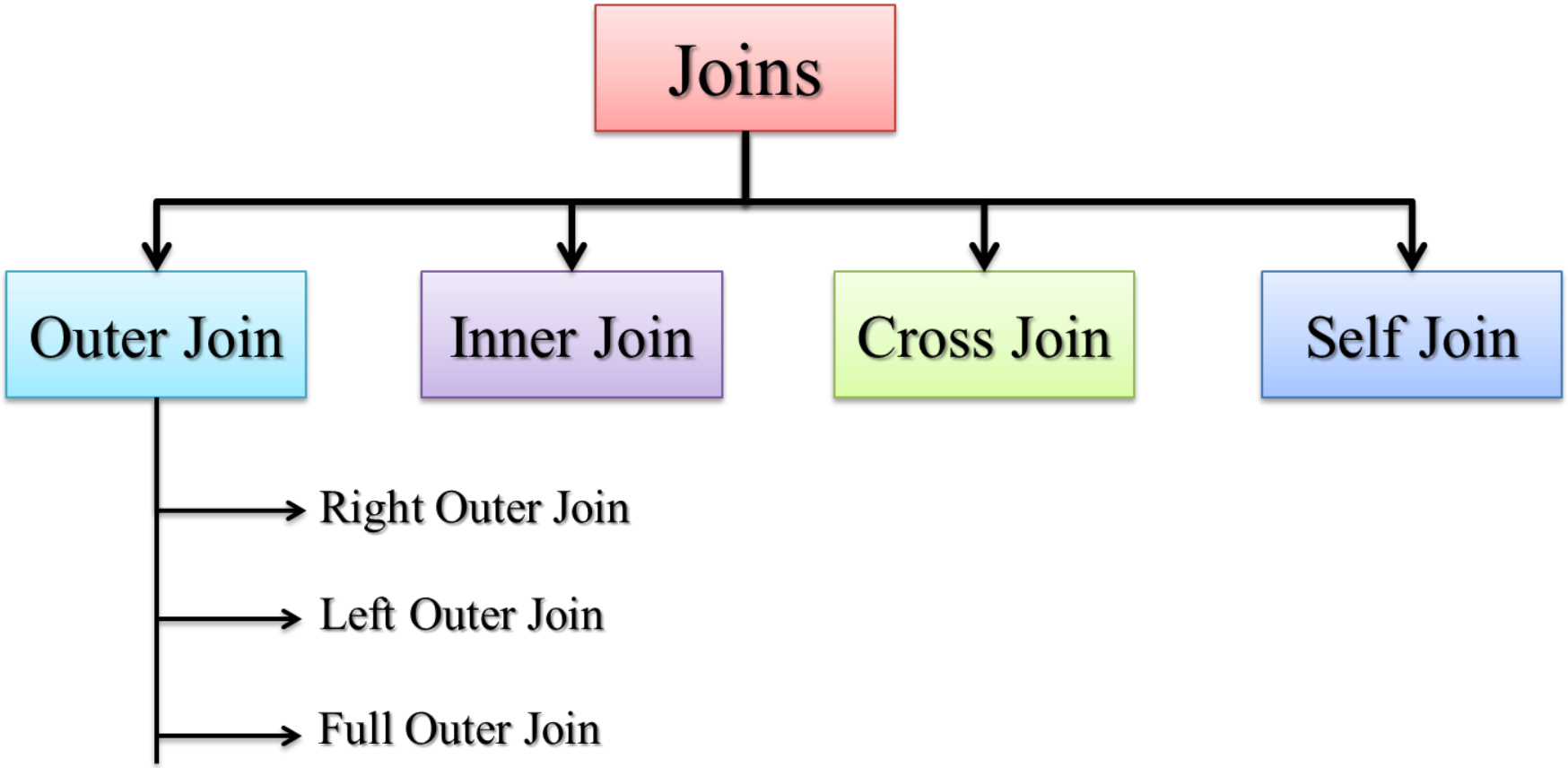
create table B
(NUM NUMBER(2),GRADE VARCHAR2(3));

- INSERT INTO A VALUES('ABC',10);
- INSERT INTO A VALUES('XYZ',11);
- INSERT INTO A VALUES('PQR',12);
- INSERT INTO A VALUES('MNO',14);
- COMMIT;

- INSERT INTO B VALUES(11,'A');
- INSERT INTO B VALUES(12,'B');
- INSERT INTO B VALUES(13,'C');
- INSERT INTO B VALUES(15,'B');
- COMMIT;

- **SELECT NUM FROM A UNION SELECT NUM FROM B;** // Displays all the records from multiple tables without duplicates.
- **SELECT NUM FROM A UNION ALL SELECT NUM FROM B;** // Displays all the records from multiple tables including duplicates.
- **SELECT NUM FROM A INTERSECT SELECT NUM FROM B;** //Displays common records from multiple tables
- **SELECT NUM FROM A MINUS SELECT NUM FROM B;** // Displays records which are not present in B
- **SELECT NUM FROM B MINUS SELECT NUM FROM A;** // Displays records from B which are note present in A

JOINS


Joins
Outer Join
Inner Join
Cross Join
Self Join
Right Outer Join
Left Outer Join
Full Outer Join

tab1

NUMID
12
14
10
11

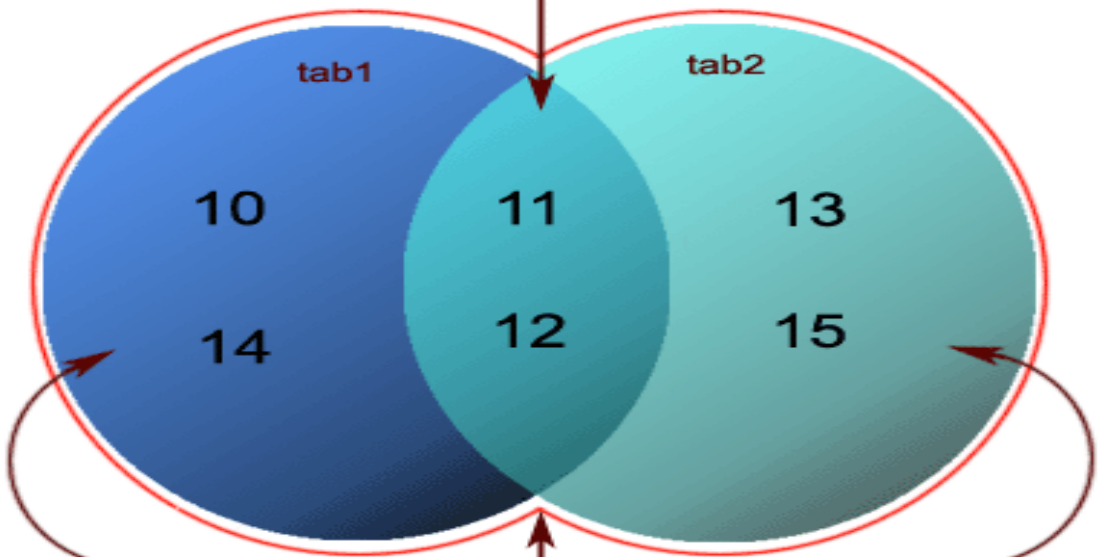
INNER JOIN

```
select * from tab1
inner join tab2
on tab1.numid=tab2.numid;
```

Result : [11,12]

tab2

NUMID
13
15
11
12




```
select * from tab1
left outer join tab2
on tab1.numid=tab2.numid;
```

Result : [10,14,11,12]

LEFT OUTER JOIN

```
select * from tab1
right outer join tab2
on tab1.numid=tab2.numid;
```

Result : [11,12,13,15]

RIGHT OUTER JOIN

```
select * from tab1
full outer join tab2
on tab1.numid=tab2.numid;
```

Result : [10,14,11,12,13,15]

FULL OUTER JOIN

JOIN QUERIES

- **//INNER/Equi Join**

SELECT * FROM A INNER JOIN B ON A.NUM=B.NUM;

- **//RIGHT OUTER JOIN**

SELECT * FROM A RIGHT OUTER JOIN B ON A.NUM=B.NUM;

- **//LEFT OUTER JOIN**

SELECT * FROM A LEFT OUTER JOIN B ON A.NUM=B.NUM;

- **//FULL OUTER JOIN**

SELECT * FROM A FULL OUTER JOIN B ON A.NUM=B.NUM;

Equi Join/Inner Join

- **Equi Join - Only matched records in both the tables**
- **Format-1**
- SELECT EMPNO,ENAME,JOB,d.DEPTNO,DNAME,LOC FROM EMP e,DEPT d WHERE e.DEPTNO=d.DEPTNO;

- **Format-2**
- SELECT EMPNO,ENAME,JOB,d.DEPTNO,DNAME,LOC FROM EMP e JOIN DEPT d ON (e.DEPTNO=d.DEPTNO);

Right Outer Join

- **Right Outer Join- Matched records+ unmatched from right table emp**
- **Format-1**
- SELECT EMPNO,ENAME,JOB,d.DEPTNO,DNAME,LOC FROM EMP e,DEPT d WHERE e.DEPTNO(+)=d.DEPTNO;
- **Format-2**
- SELECT EMPNO,ENAME,JOB,d.DEPTNO,DNAME,LOC FROM EMP e RIGHT OUTER JOIN DEPT d ON(e.DEPTNO=d.DEPTNO);

Left Outer Join-Matched

- **Left Outer Join-Matched records+ unmatched from left table dept**
- **Format-1**
- SELECT EMPNO,ENAME,JOB,d.DEPTNO,DNAME,LOC FROM EMP e,DEPT d WHERE e.DEPTNO=d.DEPTNO(+);
- **Format-2**
- SELECT EMPNO,ENAME,JOB,d.DEPTNO,DNAME,LOC FROM EMP e LEFT OUTER JOIN DEPT d ON(e.DEPTNO=d.DEPTNO);

Full Outer Join

- **Full Outer Join- Matched records + un matched records from both the tables**
- SELECT EMPNO,ENAME,JOB,d.DEPTNO,DNAME,LOC FROM EMP e FULL OUTER JOIN DEPT d ON(e.DEPTNO=d.DEPTNO);

Self Join

- Self Join: Join with a table with the same table

- SELECT E.EMPNO,E.ENAME,E.JOB,M.ENAME FROM EMP E, EMP M WHERE E.MGR=M.EMPNO;

SUB QUERIES

- Sub Query is a Query within a Query.
- Sub Query contains 2 parts.
  1. **Outer Query**
  2. **Inner Query**
- The output of inner query is become input of outer query.
- 2 Types of Sub Queries:
  1. Single row sub query

     <, >, <=, >=, !=
  2. Multi row Sub Query.

     IN,ANY,ALL

Queries...

- 1) **Display employees whose salary is less than the of ALLEN**
- SELECT * FROM EMP WHERE SAL<(SELECT SAL FROM EMP WHERE ENAME='ALLEN');
- **2) 2nd max salary from employee**
- SELECT MAX(SAL) FROM EMP WHERE SAL<(SELECT MAX(SAL) FROM EMP);

- **3) 3rd Maximum salary**

```
SELECT MAX(SAL) FROM EMP WHERE SAL<
(SELECT MAX(SAL) FROM EMP WHERE SAL<(SELECT MAX(SAL) FROM EMP));
```

- **4)Find the salary of employees whose salary is greater than the salary of employee whose EMPNO 7788.**

```
select sal from emp where sal>(select sal from emp where empno=7788);
```

- **5) display the employees who all are earning the highest salary.**

```
select * from emp where sal=(select max(sal) from emp);
```

Multi row sub queries

- **6) Display employees whose salary is equal to the salary of the atleast one employee in departmentID 10.**

```
select * from emp where sal IN (select sal from emp where deptno=10);
```

- **7)Display employees whose salary is equal to the salary of the one employee in departmentID 10.**

```
select * from emp where sal IN (select sal from emp where deptno=10);
```

- **8)Display the employees whose slary is greater than the atleast on employee in dept 10;**

```
select * from emp where sal>ANY(select sal from emp where deptno=10);
```

- **9)Display employees whose salary is less than the salary of all employees in dept 10;**

select * from emp where sal<ALL(select sal from emp where deptno=10);

- **10) Query to get department name of the employee.**
- SELECT ENAME,EMPNO,DEPTNO,(SELECT DNAME FROM DEPT WHERE EMP.DEPTNO=DEPT.DEPTNO)DNAME FROM EMP;

- **11) List out the employees who are having salary less than the maximum salary and also having hiredate greater than the hiredate of an employee who is having maximum salary.**

```
SELECT EMPNO,ENAME SAL,HIREDATE FROM EMP WHERE
SAL<(SELECT MAX(SAL) FROM EMP)
AND
Hiredate>(select hiredate from emp where Sal=(select max(sal) from emp));
```

PSEUDO COLUMNS

- ROWNUM
- ROWID

ROWNUM

- ROWNUM is pseudo column which starts with one and incremented by 1.
- Rownum values are temporary.
- Rownum values generation started from one incremented by1.
- **Examples:**
- **SELECT * FROM EMP WHERE ROWNUM<=3;** // Display first 3 rows
- **SELECT * FROM EMP WHERE ROWNUM<=6;** // Display first 6 rows
- **SELECT * FROM EMP WHERE ROWNUM<=7 MINUS SELECT * FROM EMP WHERE ROWNUM<=2;** // display 3 to 7 rows

ROWID

- ROWID is pseudo column which contains hexadecimal values.
- ROWID indicates the address where the row is stored in the database.
- ROWID values are permanent.

- Example:
- **select rowid,empno,ename from emp;**

TCL Commands

- COMMIT
- ROLLBACK
- SAVEPOINT

- These commands used with DML Commands(Insert, Update, delete)

- insert into stu values(101,'abc');
- insert into stu values(102,'abc');
- insert into stu values(103,'abc');
- savepoint s1;
- insert into stu values(104,'abc');
- insert into stu values(105,'abc');
- savepoint s2;
- insert into stu values(106,'abc');
- savepoint s3;
- select * from stu;
- rollback to s1;
- rollback;

Database User creation

- DBA user only can create a new user.
- CREATE USER KIRAN IDENTIFIED BY KIRAN123; // User creation
- GRANT CONNECT,RESOURCE to KIRAN;

- TWO TYPES OF PREVILIGES/ACCESS
- 1) System privileges(DBA)
- CONNECT,RESOURCE
- 2) Object privileges (Any user can give privileges)
- select, update, delete, insert

DCL Commands

- GRANT
- REVOKE

- Grant command is used to give privileges to the users.
- Revoke command is used to remove the privileges from the users.

- **HR USER**
- SELECT * FROM STUDENT;
- GRANT SELECT ON STUDENT TO KIRAN; // PROVIDES SELECT PREVILIGE TO THE USER KIRAN
- GRANT INSERT ON STUDENT TO KIRAN; // Provides INSERT PREVILIGE TO THE USER KIRAN
- GRANT UPDATE ON STUDENT TO KIRAN; // Provides UPDATE PREVILIGE TO THE USER KIRAN
- GRANT DELETE ON STUDENT TO KIRAN;
- REVOKE SELECT ON STUDENT FROM KIRAN; // REMOVE THE SELECT PREVILIEGE FROMKIRAN ON STUDENT
- GRANT ALL ON STUDENT TO KIRAN; // PROVIDES SELECT, UPDATE, INSERT, DELETE to the User Kiran
- REVOKE ALL ON STUDENT FROM KIRAN; // PROVIDES SELECT, UPDATE, INSERT, DELETE to the User Kiran

- **KIRAN**
- SELECT * FROM HR.STUDENT;
- INSERT INTO HR.STUDENT VALUES('ravi',115,'B');
- UPDATE HR.STUDENT SET SNAME='RAJ' WHERE SNO=115;
- DELETE FROM HR.STUDENT WHERE SNO=111;

Views

- View is logical representation of data from one or more tables.
- View does not contain any data.
- View does not consume memory location.
- When we write select statement on view , we get the data from the table.
- Tables which are used for creating the view are called a base tables.
- We can perform DML operations on simple views.(operations reflected on base table)

Types of Views

1) Simple views
2) Complex views
3) Read only views

Simple View

- When view is created using one table it is called simple view.

- CREATE VIEW EMP_V1 AS SELECT EMPNO,ENAME,JOB FROM EMP;
- SELECT * FROM EMP_V1;

Complex view

- When a view is created using multiple base tables is called complex view.
- CREATE VIEW V1 AS SELECT EMPNO,ENAME,SAL,EMP.DEPTNO,DNAME,LOC FROM EMP,DEPT WHERE EMP.DEPTNO=DEPT.DEPTNO;
- SELECT * FROM V1;
- INSERT INTO V1 VALUES(1112,'RAVI',50000,10,'ACCOUNTING','DALLAS'); //In valid
- **Insert operations are not allowed in complex views.**

Read Only view

- We can restrict DML operations on views by creating read only view.
- CREATE VIEW V3 AS SELECT * FROM EMP WITH READ ONLY;
- SELECT * FROM V3;
- UPDATE V3 SET ENAME='KIRAN' WHERE EMPNO=7369; // Invalid

Indexes

- Index is an object which is used for improve the performance of the select statements.

- 1. Simple Index
- 2. Composite Index

Simple Index

- When index is created on one column is it called as simple index.
- Index should be created on columns which we regularly use in where clause.
- When index is created , a separate structure is created with first column is rowid, column values.
- The row in the index will be arranged in ascending order.

- CREATE INDEX IDXSAL ON EMP(SAL); //CREATING INDEX

Composite Index

- When index is created on more than one column is called as composite index.

- CREATE INDEX IDX1 ON EMP(SAL,DEPTNO); **// index refers to 2 columns.**

- select index_name from user_indexes; **// displays all index created**

- drop index IDX1; **// dropping indexes.**

Synonyms

- It is alternate name of the object.
- Synonym is used instead of table names for the commands.

- create synonym E1 FOR EMP; // Creating Synonym

- SELECT * FROM E1; // Selecting data from Synonym

- drop synonym E1; // Dropping Synonym

- SELECT * FROM USER_SYNONYMS; // DISPLAYS LIST OF SYNONYMS U HAVE CREATED.