LAB 1 SQL Dt: 03-Jan-2022

Datatypes :

→ Character Datatype
→ Number Datatype
→ Date/Time Datatype

Character Datatype :-

(i) char(size)
(ii) Varchar(size)
(iii) Varchar2(size)
(iv) Nchar(size)
(v) Nvarchar2(size)
(vi) Raw(size)
(vii) Long
(viii) Long Raw

Number Datatype :-

(i) Number
(ii) Number(size)
(iii) Number(p,s) → scale (digit after decimal)
↳ Presition
(iv) Integer
(v) float
(vi) Decimal

Example : 8.57 → Number(3,2)
17.8 → Number(3,1)

Date Datatype :-

(i) DATE (DD-MON-YYYY) 31-oct-2000
(ii) TIMESTAMP

N.B : The character and date datatype are inserted with in single cots (' ').
The Number datatypes are inserted direct without single cots (' ').

Q. STUDENT (Rollno, Name, DOB, Course, CGPA).

→ CREATE TABLE Syntax (without constraint)

```
CREATE TABLE TABLENAME
(col1  datatype(size),
 col2  datatype(size),
  :
 coln  datatype(size));
```

→ CREATE TABLE STUDENT

```
(Rollno   Number(5),
 Name     varchar2(20),
 DOB      date,
 Course   varchar2(5),
 CGPA     Number(3,2));
```

→ TO VIEW THE TABLE DESCRIPTION:

Syntax: DESC TABLENAME;

N.B: SQL Queries are case insensitive.

→ To know the existing tables:

Syntax: SELECT * FROM TAB;

→ INSERTING DATA INTO TABLE:

N.B: The data will be inserted in the order in which the attributes are created in the table.

Syntax: INSERT INTO TABLENAME VALUES (data for col1, data for col2 ... data for coln);

④ INSERT INTO STUDENT VALUES (101, 'Prakash', '31-OCT-2000', 'MCA', 8.57); ↵

(For inserting another students data we have to write the above command again.)

Another way (faster/easier)

Syntax: INSERT INTO TABLENAME VALUES (&col1, &col2, ---, &coln);

⑤ INSERT INTO STUDENT VALUES (&Rollno, &Name, &DOB, &Course, &CGPA); ↵

/ ↵ (For data insert in row)

→ RETRIVING DATA / DATA RETRIVAL

(i) Retriving whole data from the table. (All rows and columns)

Syntax: SELECT * FROM TABLENAME; ↵

⑥ SELECT * FROM STUDENT; ↵

(ii) Retriving selected columns.

Syntax: SELECT col1, col2, ---, coln FROM TABLENAME;

⑦ SELECT Rollno, Course, CGPA FROM STUDENT;

Q. Retrive Rollno, Course, CGPA of all students.

⑧ SELECT Rollno, Course, CGPA FROM STUDENT;

(iii) Retriving selected Rows.

Synt: SELECT * FROM TABLENAME WHERE cond?;

The cond? is of the form (Attribute op values / Attribute op attributes)

* Here op is one of the operators among (=, !=, >, >=, <, <=, LIKE, IN, BETWEEN..., IS NULL, AND

* More than one cond? can be combined by AND, OR, NOT operators.

Q. Retrive the information of students who are acquiring CGPA 8.5 or above.

→ SELECT *
FROM STUDENT
WHERE CGPA >=8.5;

Q. Retrive the information of student who are getting CGPA 7.5 or less and belongs to MCA.

→ SELECT *
FROM STUDENT
WHERE CGPA <=7.5 AND Cource='MCA';

(iv) Retrive selected Rows and columns.

Synt: SELECT col1, col2, ... colk FROM TABLENAME
WHERE cond?;

Q. Retrive Rollno, Name, DOB, Cource of the Students who are getting CGPA 7.5 or less and studied in MCA.

SELECT Rollno, Name, DOB, Cource
FROM STUDENT
WHERE CGPA <= 7.5 AND Cource = 'MCA';

DI: IT Jan 22

BETWEEN AND operator:-

→ In between a rows of values are selected where upper and lower limit is checked. ~~in this operator~~

e.g. Employee (eid, ename, degn, sal, dob, gender, dno)

Q. Find the employee who are getting salary in the range 10000 to 30000.

SELECT * FROM Employee
WHERE Sal >= 10000 AND Sal <= 30000;
or WHERE Sal BETWEEN 10000 AND 30000;

IN operator

→ Multiple or.

Q. Find the employers who are working under dno 1 or 2 or 3.

SELECT * FROM EMPLOYEE

→ IN operator is used to match any value from a list in a SELECT statement.

→ This is alternatives to multiple OR statement

⇒ SELECT * FROM EMPLOYEE
WHERE dno IN (1,2,3);

eg Find the eid, ename, sal and dno of employees who are either manager or Senior manager.

⇒ SELECT eid, ename, sal, dno
FROM EMPLOYEE
WHERE desgn IN ('manager', 'Senior manager');

* LIKE operator :-

WILD card

(i) '%' → matches zero or more character.

(ii) '_' underscore

↳ matches any one character

Q. Find the employees whose ename is start with letter S.

⇒ SELECT * FROM EMPLOYEE
WHERE ename LIKE 'S%';

Q. Find the employees where second letter of their name is 'a'.

⇒ SELECT * FROM EMPLOYEE
WHERE ename LIKE '_a%';

Q. Find ~~the~~ ~~name~~ eid, ename, dsgn, sal of employees whose name ending with 'm'.

⇒ SELECT eid, ename, dsgn, sal FROM EMPLOYEE
WHERE ename LIKE '%m';

Q. Find the emps whose name contains the letter 'o'.

⇒ SELECT * FROM EMPLOYEE
WHERE ename LIKE '%o%';

④ IS NULL operator :-

e.g: If someone's dob is null then we can write:

SELECT * FROM STUDENT
WHERE DOB IS NULL;

⑤ DISTINCT keyword :-

e.g SELECT DISTINCT dsgn FROM EMP;

SELECT DISTINCT dsgn, sal FROM EMP;

The DISTINCT keyword is used to supress the duplicate values in a columns.

Syntax: SELECT DISTINCT COLUMN NAME FROM TABLE NAME;

Q. Select different designation from emp?.

⇒ SELECT DISTINCT desgn FROM EMP;

Q. Select different salary from emp.

⇒ SELECT DISTINCT sal FROM EMP;

SORTING:

Sorting is a table means specific ordering of rows based on a columns/columns.

Sort order

ASC → For ascending order
DESC → For desending order

N.B: → By default the sort order is Ascending (Asc)

Syntax: ~~SELECT */columnlist~~

SELECT columnlist/*
FROM {table name}
WHERE {Condn}
ORDER BY col/expression [ASC/DESC].

eg: ① SELECT SAL FROM EMP
ORDER BY SAL;

② SELECT SAL FROM EMP
ORDER BY SAL DESC;

③ SELECT ENAME FROM EMP
ORDER BY ENAME;

④ SELECT EID, ENAME, SAL FROM EMP
ORDER BY DNO, SAL;

eg. sort the salary. ①
eg sort the salary in descending order. ②

eg. Select eid, ename, desgn of emp ascending order of their salary,
⇒ SELECT eid, ename, desgn, sal FROM EMP
ORDER BY sal;

⑤ SELECT * FROM EMP
ORDER BY SAL;

eg. Sort the employees based on their ~~department no.~~ dno. and corresponding salary.

⇒ SELECT * FROM EMP
~~WHERE~~
ORDER BY dno, SAL;

dno
1
2
3
4

eg: Find the manager with ascending order of their sal.

⇒ SELECT * FROM EMP
WHERE desgn = 'Manager'
ORDER BY SAL;

24 JAN 2022

Column Alias

→ Alternative name to a column.

Syntax: ① SELECT colname "new-col-name" from <tblname>;
② SELECT colname as newcolname from <tblname>;
e.g. Display ename as employee_name;

~~⇒ Select ename~~

4 way ⇒ SELECT ename "employee-name" FROM EMP;

2 way ⇒ SELECT ename as employee-name FROM EMP;

→ If we don't use double-cont in 1st way then also it works.

Concatination

|| → concatination operator

e.g. Find eid, ename, sal in format- xxx having eid 101 is getting salary 25000 per month.

⇒ Select ename || 'having eid' || eid || 'is getting salary' || sal || 'per months' FROM EMP;

e.g. Find eid, ename, sal, dno in the format 101, xxx, 25000, 1.

⇒ select eid||',' || ename || ',' || sal || ',' || dno FROM EMP;

e.g. Display eid, ename, sal, dno in the format xxx having eid 101 is getting ~~salary~~ 2500 under department no. 1

⇒ SELECT ename || 'having eid' || eid || 'is getting' || sal || 'under department no' || dno FROM EMP;

DML

Insert, update, Delete

Syntax

UPDATE <Tablename>
SET col1 = newvalue, col2 = newvalue ... coln = newvalue } for columns fill.
WHERE <cond>;

e.g. Change the salary of all mngers to 41000.

```
UPDATE EMP
SET Sal = 41000
WHERE degn = 'mgr';
```

e.g. Change / update dept. of e04 to dept no 2.

```
UPDATE EMP
SET dptno = 2
WHERE eid = 'e04';
```

e.g. Update the salary of the managers of dno 2 to 4800.

```
UPDATE EMP;
SET Sal = 4800;
WHERE degn = 'mgr' AND dno = 2;
```

DELET statement

Syntax: DELETE FROM tablename
[WHERE Cond'n];

e.g. Delet the information of all employees.

```
DELETE FROM <tablename>;
```
(EMP)

e.g Delete the information of dno = 3;

```
DELET FROM EMP
WHERE dno = 3;
```

e.g. Delete all executives of dno 4;

(1) SELECT FROM EMP
WHERE dno = 4 AND degn = 'Executive';

COMMIT

After a DML statment is executed COMMIT is used to save the change by the DML statment in the database.

DELET FROM EMP
WHERE degn = 'PEON';
SQL> COMMIT
Commit complete.

ROLLBACK

After a DML statment is executed, Rollback cancels the change in the database.

i.e the ROLLBACK is taking the database to its previous state after the DML statment is executed.

eg. DELET FROM EMP
WHERE degn = 'PEON';
SQL> ROLLBACK

(Again all information of PEON is added in the database).

N.B!

After commit, Rollback can't be applied.

DDL Commands

CREATE
ALTER
DROP
TRUNCATE
RENAME
COMMENT

DROP Statment

DROP table command is useef to delet a table on the database permanently along with its structure.

Syntax: DROP TABLE <TABLENAME>;

eg. DROP TABLE EMP;

TRUNCATE Statment

~~DROP~~ Tourcate is useef to delete the data from a table. But the structure remains present/ ~~exact~~ entact.

Syntax: TRUNCATE TABLE <Tablename>;

V. B

DROP → Data along with structure of the table will be deleted permanently from the database.

TRUNCATE → Data will be deleted but the structure of the table exist in the database.

(#) Difference bet'n TRUNCATE AND DELETE

TRUNCATE	DELETE
→ DDL	→ DML
→ Data will be deleted permanently i.e Rollback can't be applied	→ data will be deleted but we can recover data using rollback if commit is not applied.

31 JAN 22

ALTER TABLE statement

↳ It is used to modify structure of a table.

i.e you can → add columns, → delete columns, add a constraint, → delete/drop a constraint, → creat or destroy indexes in database, → change the datatype of existing columns, → renaming columns or constraints.

→ Adding a new columns to an existing table.

Syntax: ALTER TABLE <tablename>
ADD (newcol1 datatype(size),
newcol2 datatype(size),
.
.
.
newcoln datatype(size));

Add a column contact no. in employee table:

→ ALTER TABLE EMP
ADD (contactno NUMBER(10));

How to fill a newly added column:

→ UPDATE <TABLEname>
SET newcolm = &newcolm
WHERE oldcolm = &oldcolumn;

→ DROP a column from a table:

Syntax: ALTER TABLE <tablename>
DROP COLUMN <column name>;

Q. DROP contact no.

→ ALTER TABLE EMP
DROP COLUMN contactno;

→ RENAMEING a column to a new column.

Syntax: ALTER TABLE <table nme>
~~RENAME COL oldcolumnme, oldname TO Newcolname;~~

RENAME COLUMN oldcolname TO Newcolname;

eg. ALTER TABLE EMP
RENAME COLUMN eid TO emp_id;

⇒ MODIFING datatype of a column.

Cond^{n}: If data is present in a column, it is not adviceable to

→ change the datatype

→ decrease the size of the datatype

SYNTAX: ALTER TABLE <TABLE NAME>
MODIFY (columnname newdatatype(size));

e.g. ALTER TABLE EMP
MODIFY (sal NUMBER(8));

⇒ CREATING a table from another table

EMP(eid, ename, degn, sal, dob, mgr_id, dno)

Emp_1 (eid, ename, degn, sal, dob, mgr_id, dno)

Emp_2 (eid, ename, sal, dno)

—Creating a table from an existing table:

SYNTAX: CREATE TABLE <tablename>
AS SELECT * FROM <existing tablename>;

eg. CREATE TABLE EMP_test1
AS SELECT * FROM EMP;

N.B: The destination table is not going to be populated with data when the cond? will be false.

or ie to copy structure of a table to a new table.

e.g: CREATE TABLE EMP_test1
AS SELECT * FROM EMP
WHERE 1=2;

→ CREATE Table with Specific columns from an existing table.

SYNTAX: CREATE TABLE (tablename) (col1, col2 ... coln)
AS SELECT col1, col2, ... coln FROM (existing tbl);

Q. (i) Creat a table EMPDEPT3 with cols eid, ename, sal, dno from employee table.
(ii) deptno = 3.

(i) CREATE TABLE EMPDEPT3 (eid, ename, sal, dno)
AS SELECT eid, ename, sal, dno FROM EMP;

(ii) CREATE TABLE EMP_DEPT3 (eid, ename, sal, dno)
AS SELECT eid, ename, sal, dno FROM EMP
WHERE dno = 3;

NOT NULL

eg.
```
CREATE TABLE EMP
(eid     NUMBER(10),
ename    VARCHAR2(30) NOT NULL,
degn     VARCHAR2(15),
SAL      NUMBER(10) NOT NULL,
DNO      NUMBER(5));
```

PRIMARY KEY DEFIND AT COLUMN LEVEL

eg.
```
CREATE TABLE EMP
(eid     NUMBER(10) Primary Key,
ename    VARCHAR2(30) NOT NULL,
degn     VARCHAR2(15),
DNo      NUMBER(5));
```

PRIMARY KEY defing at table level

eg
```
CREATE TABLE EMP
(eid     NUMBER(3),
ename    VARCHAR2(30),
degn     VARCHAR2(15),
SAL      NUMBER(10) NOT NULL,
DNO      NUMBER(3),
Primary key (eid));
```

Q. Create a table XYZ with number type attributes x_1, x_2, x_3 and make x_1, x_2 combinedly as primary key.

→
```
CREATE TABLE XYZ
(x1  Number(12),
 x2  Number(10),
 x3  Number(5),
 Primary key (x1,x2));
```

Naming a Constraint:

Syntax: constraint <constraintname> constraint def

constraint ← Keyword
constraintname ← user defined name

~~Q create table~~ eg. ~~CREATE TABLE XYZ~~
~~(x1 Number(3) constraint PK_x1~~
Primary

eg. CREATE TABLE XYZ
(x1 Number(3) constraint PK_x1 Primary key,
x2 Number(4) NOT NULL,
x3 Number(5));

Naming primary key at table level

eg Create table XYZ
(x1 Number(3),
x2 Number(4),
x3 Number(5),
constraint PK_x1x2 Primary key (x1,x2));

SELECT CONSTRAINT NAME FROM TABLE.

Syntax:

```
SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE
  FROM USER_CONSTRAINTS WHERE
  TABLE_NAME = 'XYZ';
```

(capital letter)

UNIQUE KEY :-

Dt: 21. FEB22

→ Like primary key it has the uniqueness property.

→ Like primary key the unique key does not allow duplicate values.

→ It can be of one column or more than one column.

→ Basic difference bet^n primary key & the unique key is

(i) The primary key does not allow null values but unique key allow null values.

(ii) Only one primary key is allowed to a table but more than one unique key

(For a table only one primary key is allow but there may be more than one unique key.)

(iii) Unique key are of column level

```
CREATE TABLE (TABLENAME)
(column1 datatype(size) constraint PK_col1 primary key(col1),
column2 datatype(size) Not null,
column3 datatype(size) Constraint UNIQUE_col3 UNIQUE
column4 datatype(size) );
```

UNIQUE KEY at Table level:-

```
CREATE TABLE TABLENAME
(col1  datatype(size),
 col2  datatype(size) NOT NULL,
 col3  datatype(size),
  :
 coln  datatype(size),
 CONSTRAINT PK_col1 PRIMARY KEY (col1),
 CONSTRAINT UNIQUEcol3 UNIQUE (col3));
```

Q. Create a table Emp(eid, ename, dept, sal, dob, dno) with eid as primary key. ename and dob combinedly as unique key and sal as not null and also name each of the constraint.

```
CREATE TABLE EMP
(eid   NUMBER(5),
 ename VARCHAR2(20),
 dept  VARCHAR2(20),
 SAL   NUMBER(10) Constraint NN_SAL NOT NULL
 dob   DATE,
 dno   NUMBER(2),
 CONSTRAINT PK_eid PRIMARY KEY(eid),
 CONSTRAINT UN_enamedob UNIQUE(ename,dob));
```

CHECK Constraint:

A check integrity constraint requires that a cond' to be true for a row to be produced or created.

The cond' within a check constraint is a boolean expression evaluated either true or false.

If the cond' evaluates to false an appropriate error msg will be displayed and the process is stop.

→ Business rule constraint.

Syntax:

At column level

COLUMN NAME DATATYPE(size) CHECK(exp)

Naming

COLUMNNAME DATATYPE(size) constraint ck_col check (expression)

At TABLE LEVEL

CONSTRAINT ck_col CHECK (EXPRESSION)

↓

Column name op value/ Column name

op → operator (<, >, <=, IN, AND, ...)

Q. Create a table EMP(eid, ename, degn, sal, gender, dno)
Make eid as unique and not null.
The gender take only the values M, F and
AND The dno can't be a -ve value
or a value greater than 5 or 0 value

```
CREATE TABLE EMP
(eid NUMBER(5) NOT NULL,
 ename VARCHAR2(20),
 degn VARCHAR2(10),
 SAL NUMBER(5),
 gender VARCHAR(1),
 dno NUMBER(5),
 CONSTRAINT E_id UNIQUE (eid),
 CONSTRAINT E_DNO CHECK (~~SAL BETWEEN~~
 ~~SAL > 0 AND SAL~~
 ~~SAL > 0 AND SAL <5~~)
```

```
CREATE TABLE EMP
(eid NUMBER(5),
 ename VARCHAR2(20),
 degn VARCHAR2(10),
 SAL NUMBER(5),
 gender VARCHAR2(1),
 dno NUMBER(2),
 CONSTRAINT PK_eid PRIMARY KEY(eid),
 CONSTRAINT ck_gender CHECK(gender IN(M,F,T)),
 CONSTRAINT ck_dno CHECK(dno BETWEEN 1 AND5
```

FOREIGN KEY Constraint :-

23 FEB 22

→ Referential integrity constant can be implemented by the concept of foreign key.

→ It is used to maintain consistancy among rows in table.

→ It is used to maintain rel'n among tables and rel'ns.

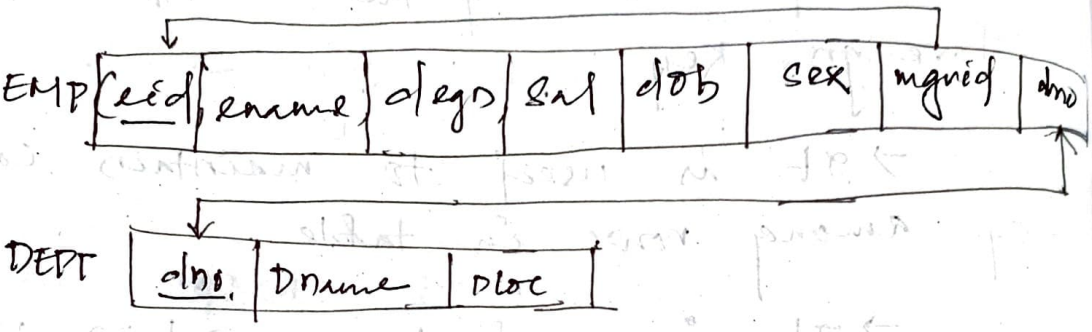
→ A foreign key is a set of attributes of column whose value is derived form primary key or unique key of some other table or same table provided the datatype and size of primary key and foreign key is same.

→ The table in which foreign key is defined is known as dependent table or child table or foreign table.

→ The table that define the primary key or unique key and is referenced by foreign key is known as referenced table or master table or Parent table.

The foreign key can be defined in the create table statement or ALTER TABLE statement after table is created.

e.g



Here dno. of ~~Emp~~ EMP table is defined as foreign key which references dno. of DEPT table (where dno. is the primary key of the DEPT table)

In the EMP table Mgrid is the FOREIGN key which is referencing to the eid of the same table EMP.

Syntax:

FOREIGN KEY AT COLUMN LEVEL

COLUMN NAME datatype(Size) REFERENCES <tablename(colm

AT TABLE LEVEL

eg

FOREIGN KEY (COLUMN Name) REFERENCES <tablename (colnm)

Naming FOREIGN key

e.g.
CONSTRINT FK_Dno FOREIGN KEY(dno) REFERENCES DEPT(dno),

eg. Make dno as foreign key in EMP table which references dno. in DEPT TABLE.

① CREATE TABLE DEPT
(dno Number(2),
dname Varchar2(20),
dloc Varchar2(20),
CONSTRAINT PK_Dno PRIMARY KEY(dno));

CREATE TABLE EMP
(eno Number(2),
ename Varchar2(20),
degn V
sal
dob
sex Varchar2(1),
mgrid Number(4),
dno Number(2),
CONSTRAINT PK_eno PRIMARY KEY(eno),
CONSTRAINT FK_dno. FOREIGN KEY(dno) REFERENCES
DEPT(dno));

Q. Make Mgrid as foreign key on EMP which references eid in same table EMP.

→

```
CREATE TABLE EMP
(EID NUMBER(10),
 ENAME VARCHAR(20),
 ELOC VARCHAR(10),
 MGRID NUMBER(10),
CONSTRAINT PK_ID PRIMARY KEY(EID),
CONSTRAINT FK_MID FOREIGN KEY(MGRID) REFERENCES
                                    EMP(EID));
```

28 FEB 22

DEFAULT Behavior of FOREIGN KEY

(i) The records in the child table can't be inserted if the corresponding records in the master table doesn't exist.

(ii) Records in the master table can't be deleted if corresponding records in child table actully exist.

→ The default behavior of foreign key can be changed by using

(i) ON DELETE CASCADE
(ii) ON DELETE ~~SETNULL~~ SET NULL

(i) ~~ON~~ DELETE CASCADE :

If ON DELETE CASCADE option is set a delete operation in the master table will trigger a delete operation for corresponding records in all child tables.

Syntax : ~~ON DELETE CASCADE~~

eg: CONSTRAINT FK_dno. FOREIGN KEY(dno)
REFERENCES DEPT(dno) ON DELETE CASCADE));

(c) ON DELETE ~~SET NULL~~ SET NULL option:

If ON DELETE SET NULL is set & delete operation in the master table will set the value held by foreign key of the child table as NULL.

eg.

ALTER TABLE COMMAND TO ADD OR DROP A CONSTRAINT:

→ To add a constraint

Syntax: ALTER TABLE <TABLE NAME>
ADD ~~CONSTRAINT def~~
Constraint_def?

eg. ① SUB (Subco, credit), we have to ~~enforce~~ set subco as Primary key.

⇒ ALTER TABLE SUB
ADD CONSTRAINT ~~PK~~_code PRIMARY KEY (Subco);

② INSTRUCTURE (cid, name, Subco)

⇒ ALTER TABLE INSTRUCTURE
ADD CONSTRAINT PK_id PRIMARY KEY (cid);

SET FOREIGN KEY :

ALTER TABLE INSTRUCTURE
ADD CONSTRAINT FK_subco FOREIGN KEY (subco)
REFERENCES SUB (subco) ;

→ TO DROP a constraint :

Syntax : ALTER TABLE (TABLE_NAME)
DROP CONSTRAINT CONSTRAINT_NAME ;

Is → ALTER TABLE INSTRUCTURE
DROP CONSTRAINT FK_subco ;

eg. ALTER TABLE SUB
DROP CONSTRAINT PK_subco ;

7 MAR 2000

eg. EMP (eid, ename, degn, dob, dno)
DEPT (dno, dname, dloc)

Find the employees who works research department.

Soln :

SELECT e.eid, e.ename, e.degn, e.dob,
e.dno, d.dname
FROM EMP e JOIN DEPT d
ON EMP.dno = DEPT.dno
WHERE d.dname = "Research" ;

JOINS in ORACLE :

Join is a query that is used to combine rows from two or more tables or views so it retrives data from multiple table/views and creat a new table or views as resultant.

JOIN CONDITIONS :-

There may be atleast one join condition in the form clause or in the where clause for joing two tables.

It compairs two columns from different tables and combines pair of rows each containing one row from each table for which the join condition is true.

The datatype and size of Joined attribute should be same.

All the operators are used in Join conditions are $>=$, $<$, $>$, $=$, $!=$, $<=$

And more than one condition can be joined by AND, OR, NOT operator.

- INNER JOIN
- EQUI JOIN
- SELF JOIN
- NATURAL JOIN
- CROSS JOIN
- OUTER JOIN
  - left outer Join
  - Right outer Join
  - Full outer Join

- Anti Join
- Semi Join

e.g. EMP(eid, ename, dno)
DEPT (dno, dname)

old style Join :

Syntax: SELECT COLUMNS
FROM TABLE1, TABLE2
WHERE TABLE1.x = Table2.x;

or SELECT

Cross Join (old style)

SELECT columns cols

FROM TABLE 1, TABLE 2;

eg. SELECT * FROM EMP, DEPT;

JOIN OR INNER JOIN

Syntax: SELECT COLUMNS

FROM TABLE1 T_1 JOIN TABLE2 T_2

ON $T_1.X$ op $T_2.X$;

(op is one of the operator >, >=, <, <=, =, !=)

Syntax: SELECT COLUMNS

FROM TABLE1 T_1 INNER JOIN TABLE2 T_2

ON $T_1.X$ op $T_2.X$;

USING

SELECT column

FROM TABLE1 T_1 JOIN TABLE2 T_2

USING (X);

EQUI JOIN

Syntax: SELECT COLUMNS

FROM TABLE1 T_1 JOIN TABLE2 T_2
ON $T_1.X = T_2.X$;

NATURAL JOIN

Syntax: SELECT COLUMNS/*

FROM TABLE1 T1 NATURAL JOIN
TABLE2 T2 ;

SELF JOIN

~~ek esa~~ Joining ~~Joing~~ a table with itself.

Syntax: SELECT COLUMNS
FROM TABLE1 T_1 JOIN TABLE1 T_2
ON T_1.Joining attri = T_2.Joining attri ;

CROSS JOIN / CROSS PRODUCT

SELECT COLUMNS
FROM TABLE1 T_1 CROSS JOIN TABLE2 T_2 ;

ASSIGNMENT- 4

CREATE 3 TABLES SAILOR, BOAT, RESERV

Convert all those given relational algebric queries into SQL except division.

① Find the name of the dept where Sanil works.

② Find the employees who works in MNSR. display 10%.

③ Increase the salary of employees who are working in MNSR.

④ Find the employees who are working under project Banking System.

⑤ For the projects located in Hyderabad find employee details with department details.

Outer Joins

Dt: 14 MAR 22

Q) Find the employee name and their corresponding manager name.

Select e.ename, e.mgrname

FROM EMP(ename, mgrid, degn)

MGR(mgrid, mgrname)

```
Select e.ename, m.mgrname
FROM   EMP e JOIN MGR m
  ON   EMP.mgrid = MGR.mgrid
WHERE  e.degn = 'MANAGER';
```

Emp(eid, ename, mgrid)

eid ename mgrid

EMP

eid	ename	mgrid
e01	xxx	e02
e02	yyy	e02
e03	zzz	e04
e04	ppp	e02

e1.eid	e1.ename	e1.mgreid
e01	xxx	e02
e01	xxx	e02
e02	yyy	e02
e02	yyy	e02

e2.eid	e2.ename	mgreid
e01	xxx	e02
e02	yyy	e02
e01	xxx	e02
e02	yyy	e02

```
SELECT e1.ename "emp_name",
       e2.ename "mgr_name"
from emp e1 join emp e2
on e1.mgreid = e2.eid;
```

Outer Joins:

An outer join is similar to equi join but it also gets the non-matched rows/tuples from the tables.

It is categorized as

(i) Left-outer Join
(ii) Right-outer Join
(iii) Full-outer Join

(i) Left - outer JOIN

It returns all the rows of the left hand side of the first table specified in the ON condition and only those rows from the right table (2nd table) where the join cond'n is made.

Syntax:

```
SELECT cols
FROM Table T1 LEFT [outer] JOIN
                    Table T2
ON T1.col = T2.col ;
```

Left outer join

Old Syntax:

```
SELECT cols
FROM TABLE1 T1, TABLE2 T2
WHERE T1.col = T2.col(+);
```

(ii) Right - Outer Join

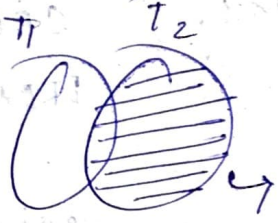
It returns all the rows from the right-hand side table specified in the on condition and only those rows from the other table where the join condition is made.

Old syntax:

Select cols
from table1 T_1, Table2 T_2

where

Syntax: SELECT cols
FROM TABLE T_1 RIGHT JOIN TABLE T_2
ON T_1.col = T_2.col;



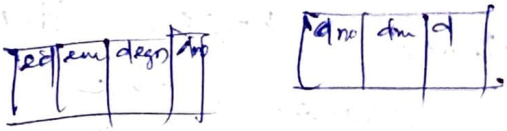
↳ Right outer Join

(iii) FULL OUTER JOIN:

Full outer join returns rows from both the tables...

So it places null where the join cond? is not mate..

Syntax: SELECT cols
FROM TABLE T_1 FULL JOIN TABLE
ON T_1.col = T_2.col;



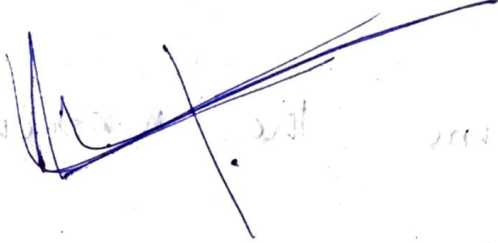
SELECTS *dname
FROM emp LEFT JOIN emp
ON emp.dno = DEPT.do
WHERE

SELECT e.ename, e.deptno

FROM ~~EMP e, DEPT d~~ Dept d ~~EMP e~~ left JOIN ~~DEPT d~~ emp e

ON EMP.deptno = DEPT.deptno

WHERE e.deptno is NULL;



Assignment Related:- 23 MAR 22

Aggregate Functions / Group functions:-

AVG
MIN
MAX
SUM
COUNT

AVG()

Returns an average of n values ignoring NULL values in a column.

Syntax: ~~avg~~ AVG(col/expr).

// Find average salary of employees.

⇒ SELECT AVG(SAL) FROM EMP;

MIN()

Returns a minimum value.

// Find the minimum salary from the employee table.

⇒ SELECT MIN(SAL) FROM EMP;

MAX ( )

Returns the maximum value.

e.g. SELECT MAX(SAL) FROM EMP;

SUM ( )

Returns the sum of the values.

// Find the monthly expenditure encurred by the company towards employee Salary.

⇒ SELECT SUM(SAL) FROM EMP;

COUNT ( )

It returns the number of rows.

COUNT (expr)

Returns no. of rows where expr is not to null.

COUNT (*)

It returns the no. of rows in the table including duplicate and those with nulls.

// Find the total no. of employees in the company:

(1) SELECT COUNT(*) FROM EMP;

N.B: These aggregate/group functions can be applied on specific groups, so the groups are made by the ~~group by~~ GROUP BY clause in select statements.

If GROUP BY clause is not specified, so all the records in a table can be taken as a group and the group functions are applied accordingly.

GROUPING OF DATA :-

GROUP BY CLUSE

In oracle GROUP BY clause is

used in select statement to collect data from multiple records and group the results by one or more columns.

(ii) GROUP BY clause is used in a table to devide the rows in different groups and the groups are treated separetly.

(iii) Group functions can be applied into individual groups.

(iv) HAVING clause is used with the group by clause to restrict the groups.

WHERE CLUSE & HAVING CLUSE :-

WHERE clause restrict the rows where as HAVING clause restrict the group.

Q/ Make different group of employees according to their salary.

A) SELECT SAL FROM EMP GROUP BY (SAL);

Q/ Make different group of employees according to their department name.

A)

Q/ Find the department no. where 3 or more than 3 employee are working.

A) SELECT DNO FROM EMP GROUP BY(DNO)
HAVING COUNT(EID) >= 3;

Q// Make different group of employee acc to there dept. no. And count the no of employee of each group.

(7) SELECT DNO, COUNT(EMPNO) "no-of-emp"
FROM EMP GROUP BY (DNO);

Q// Find the avarage salary in each department.

Q// Find the maxm sal in each department.

Q// Find the minm sal of each department.

Q// Find the total sal of each dept for month.

(8) Q// find avrage no. of employee of a department. (SELECT AVG(COUNT(EMPNO))

Q// Find the dept no. where avrage sal > 15000.

Q// Find the dept. where maxm no. of employee are working.

Q// Find the dept. where minm no. of employees are working.

// Group the employee according to their salary. The cond? is grouping is done above 7000 salary (and each group should contain atleast 3 employee).

⇒
```
SELECT EID FROM EMP
WHERE SAL > 7000
GROUP BY (SAL)
HAVING SAL > 7000 AND HAVING
COUNT (EID) >= 3;
```

* We can't use where clause after group by but we can use before it.

SET

SET operations in Oracle :-

UNION
UNION ALL
INTERSECT
MINUS

The set operations are used to join the results of two (or more) Select statments.

Syntax: SELECT QUERY 1
{set operation}
SELECT Query 2 ;

The select Query 1 and select Query 2 must be union compatible i.e no. of columns and the datatype and size of corresponding columns in both the select queries must be same to apply the set operation.

UNION -

It returns the combine result of the two select statment. So it will list only one record out of the duplicate records.

UNION ALL :-

It is like the operation UNION but it returns the duplicate records from the final result.

INTERSECT :-

INTERSECT lists only the records that are common to the select queries.

MINUS :-

It returns only the records which are resulted only from the select query 1 but not from the select query 2.

i.e the MINUS set operator removes the second query's result from the output if their also found in the 1st query's result.

e.g. STUDENT (sid, sname, cource, section)

BOOKS (bid, bname, sid)

1. Find the student who have not issued any books yet.
2. Find the books which are not obtained by any students yet.
3. Find the students who are

4. Find the students who have issued both DBE books and economic book.

5. Find the students who are obtained algorithm book or DS book.

6. Find the students who are issued History book but not economic book.

7. Group the book according to their name and find the no. of students who are obtained for these books

8. Find the no. of students admitted for each course.

9. Find the no. of students present in each section of this course.

SUB QUERY :-

28.MAR.22

⇒ It is also known as "nested Query". It is usually a select query in another select query.

⇒ A subquery can be used within where, ~~HAVING~~ or FROM clause from another query.

The general form of subquery

→ outer query

SELECT QUERY

op (select query);

→ Inner query

In sub query inner query will be executed first and returns value(s) to the outer query. The outer query will be executed next with the result from the inner query.

It is of 2 types

(i) Single row subquery

(ii) Multiple row subquery

(i) Single row Subquery:

A subquery that returns only one row of data.

It is also known as Scalar subquery.

Operators which are used in the ~~Single row~~ outer query and Conditional operators

>=, <=, =, !=

_ N.B! _

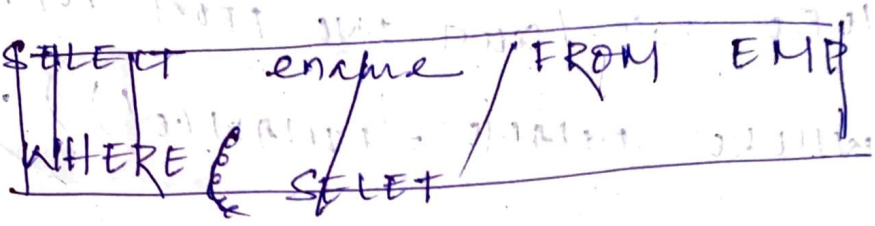
→ The subquery must be enclosed with a pair of Parenthesis.

→ ~~The~~ subquery returns some columns in most of the cases so if nothing is return NULL value will be return from the subquery.

→ Subquery is used in R.H.S of Conditional and Relational operator.

→ Order by clause can't be used in a ~~table~~ subquery.

Q// Find the name of department where JohnSmith work.



→ SELECT dname of department WHERE dno = (SELECT dno FROM EMP WHERE ename = 'JohnSmith');

Q// Find the employees who are getting more salary than John.

Q// Find the employees who don't works in the dept where John work.

Q// Find the employees who are getting less salary than the minimum salary of finance dept.

Q// Find the employees who are getting more salary than the average salary of ~~dept~~ Research department.

Q/ DELET the information of marketing team.

(3) SELECT ENAME FROM EMP
WHERE SAL < (SELECT MIN(SAL) FROM EMP
WHERE DNO = (SELECT DNO FROM DEPT
WHERE DNAME = 'FINANCE'));

3-APR 22

Multiple Row Subquery :-

Q/ Find the eid, ename of employees who are working under resource or marketing or finance Department.

SQL) SELECT EID, ENAM FROM EMP
WHERE DNO IN (SELECT DNO FROM DEPT
WHERE DNAME IN ('Resource', 'Marketing', 'Finance'));

Def':
SE returns more than one row or values.

The operators used in single row Subquery can't be used in multiple row Subquery.

The operators are used in multiple row subqueries are

① IN - Compare with / equals to any values in the list.

② ANY or SOME - Compare the given value with the value returned by the subquery.

③ ALL - Compare the given value with every value returned by the subquery.

ANY operator / SOME operator:

→ It is used in combination with other relational operators <, >, =

→ It works like OR operator.

*) <ANY - less than the maximum value in the list.

*) >ANY - Greater than the minimum value in the list.

*) =ANY - Equal to any value in the list.

Q// Find the minimum SAL of Finance department:

SQL> SELECT MIN(SAL), FROM EMP
WHERE DNO=(SELECT DNO
FROM DEPT WHERE DNAME='Finance'));

Q// Select the employees who are getting less salary than that of finance in HR department.

SQL> SELECT SAL FROM EMP
WHERE SAL < (SELECT MIN(SAL)
FROM EMP WHERE DNO
IN (SELECT DNO FROM DEPT WHERE
DNAME IN ('FINANCE', 'HR'));

ALL Operator :

ALL operators are used with relational operators like >, <, =.

>ALL ⇒ More than the max^m value.

<ALL ⇒ Less than the min value

=ALL ⇒ ~~equal to any value~~ meaningless / don't use

Q// Find employees who are getting less than ~~the~~ ~~salary~~ the average salary of each dept.

Q// Find the dept no. where maximum no. of employees are working.

SAL

A) SELECT * FROM EMP
WHERE SAL < (SELECT AVG (SAL)