DBMS

Logical DBMS Architecture or Three level Architecture of DBMS:

* The External or view level
* The conceptual or Global level
* The internal or physical level

Data modelling is the process of documenting the logical design in the form of an easily understood diagram, using text and symbols to represent the way data needs to flow.

Ex: Entity Relationship Modelling (ER diagram)

DBMS types:

* Hierarchical database: tree datastructure
* Network database
* Object relational database
* Relational database

**Schema:** The overall design or description of the database is the database schema.

* Physical Schema
* Logical Schema

**Keys:**

* Super Key - **any combination** of fields that **uniquely identify a row**.
* Candidate Key - **least combination** (1 or more) of fields that **uniquely identify** **a row.**
* Primary Key - most **appropriate Candidate Key.**
* Foreign Key - **references** values from **primary key of another table**.

**SQL categories:**

* Data Definition Language (DDL)
* Data Manipulation Language (DML)
* Transaction Control Language (TCL)
* Data Query Language (DQL)

**Data Definition Language (DDL):**

* CREATE
* ALTER
* DROP
* RENAME
* TRUNCATE

**Data Types:**

* CHAR (size)
* VARCHAR2 (size)
* NUMBER (size, precision)
* DATE
* LONG

**CREATE:**

CREATE TABLE table\_name Create Table Employee

( (

Column\_name1 DATATYPE(Size), Emp\_id number(4) NOT NULL,

Column\_name2 DATATYPE(Size), Name varchar2(20) NOT NULL,

Column\_name3 DATATYPE(Size), Salary number(8),

); );

**ALTER:**

**Add -** ALTER TABLE emp **ADD** age number(3) not null;

Modify - ALTER TABLE emp **MODIFY** salary number(10,2);

Rename- ALTER TABLE emp **RENAME COLUMN** salary **to** em\_sal;

Drop - ALTER TABLE emp **DROP COLUMN** age;

**TRUNCATE:**

TRUNCATE TABLE Employee; - remove all rows and resets schema

**DROP:**

DROP TABLE Employee; - completely deletes the table

**RENAME: (**only command used **without TABLE)**

**RENAME** Employee to Emp\_table;

**Data Query Language (DQL): only SELECT**

**SELECT** column\_name1, column\_name2…..

**FROM** table\_name

[**WHERE** search\_conditions ]

[**GROUP BY** grouping\_column\_name ]

[**HAVING** aggregated\_search\_condition ]

[**ORDER BY** sorting\_column]

**Restricting Selection using comparison operators:**

SELECT name, age FROM customer\_details WHERE salary>50000;

**Restricting Selection using IN keyword:**

SELECT name,location FROM customer\_details WHERE location IN ('Kolkata','Delhi');

**Restrict based on pattern matching:**

SELECT name,location FROM customer\_details WHERE name LIKE '\_a%';

The above query displays name and location of customers whose **name's second character is a.**

**Restricting Selection using BETWEEN …**

SELECT name,location FROM customer\_details WHERE age BETWEEN 20AND 40;

**Restricting based on NULL value:**

Condition on NULL value can be given using the keyword IS.

**SELECT name, age FROM customer\_details WHERE location IS NULL;**

The above query displays name and age of customers whose location not specified

**SELECT name, age FROM customer\_details WHERE age IS NOT NULL;**

The above query displays name and age of customers whose age available

**Order By:**

Select name, age, salary from customer\_details where salary > 50000 **order by salary**

// use DESC for descending

**Group By:**

Select count(name), age from customer\_details **group by age;**

**Data Manipulation Language:**

* **Insert**
* **Update**
* **Delete**

**Insert:**

**INSERT INTO** table\_name (column1, column2, **…) VALUES** (value1, value2,…….)

INSERT INTO table\_name VALUES (value1, value2,……..) //if all columns are there

Using &value1, & value2 makes it interactive

Enter value for value1: “agegag”

**Update:**

**UPDATE** table\_name **SET** column\_name = value1, ….. **where condition**;

Update phone\_book set city = “Mumbai” where name= “Raj” ;

**Delete:**

**DELETE FROM table\_name where condition;**

Delete from phone\_book where phone\_no=9523224423

**Transaction Control Language:**

* Commit
* Savepoint
* Rollback

Commit - makes data changes permanent

Savepoint - to control reverting of changes

Rollback - data changes are undone and previous state of data is restored

Ex:

INSERT INTO employee VALUES(10,'JHON',3000);  
INSERT INTO employee VALUES(10,'KELLY',2000);  
**SAVEPOINT S1;**  
INSERT INTO employee VALUES(10,'WILSON',4000);  
**ROLLBACK TO S1;**

**Constraint Types:**

* **PRIMARY KEY //**cannot be LONG, LOB datatype
* **FOREIGN KEY**
* **UNIQUE**
* **NOT NULL**
* **CHECK**
* **DEFAULT**
* **Composite PRIMARY KEY** can have max 32 columns and need to define only at table level.

**Inline or column level declaration:**

Ex: CREATE TABLE Location

(

loc\_id **NUMBER(4**) CONSTRAINT loc\_pk PRIMARY KEY,

street\_address **VARCHAR2(20),**

city **VARCHAR2(15),**

state **VARCHAR2(15),**

pincode **NUMBER(6)**

);

**Composite PRIMARY KEY at Table level:**

CREATE TABLE sales

(

sales\_id **NUMBER(6),**

cust\_id **NUMBER(4),**

prod\_id **NUMBER(5),**

qnty **NUMBER(6),**

sold\_date **DATE,**

CONSTRAINT sales\_pk PRIMARY KEY(sales\_id, cust\_id, prod\_id)

);

**Primary key with Alter statement:**

ALTER TABLE person ADD CONSTRAINT aadhar\_pk PRIMARY KEY (aadhar\_id);

**Removing Primary Key:**

ALTER TABLE location DROP CONSTRAINT loc\_pk;

**Foreign key constraint is called the Referential Integrity Constraint.**

**Foreign key with Create statement -Column Level:**

CREATE TABLE person

(

name **VARCHAR2(30),**

dob **DATE,**

gender **CHAR(1),**

address **NUMBER(4)** REFERENCES location(loc\_id),

gender **CHAR(1)**

//CONSTRAINT address\_fk FOREIGN KEY(address) REFERENCES location(loc\_id**) table level**

**Optional part |^|**

);

**Foreign key with Create statement -Table Level //**composite

CREATE TABLE production

(

suppl\_id **NUMBER(4),**

warehouse\_id **NUMBER(6),**

warehouse\_name **VARCHAR2(20),**

product\_name **VARCHAR2(20),**

qnty **NUMBER(4),**

unit\_price **NUMBER(6,2),**

**CONSTRAINT** suppl\_fk **FOREIGN KEY**(suppl\_id) **REFERENCES** supplied(supp\_id),

**CONSTRAINT** warehouse\_fk(warehouse\_id, warehouse\_name) **REFERENCES** (id,name)

);

ALTER TABLE person ADD CONSTRAINT location\_fk FOREIGN KEY(address) REFERENCES location(loc\_id) ON DELETE NULL OR ON DELETE CASCADE; // part of below sentences

ALTER TABLE person DROP CONSTRAINT location\_fk;

**ON DELETE SET NULL:**

A foreign key with ON DELETE SET NULL means , if a record in the parent table is deleted then the corresponding records in the child table will be automatically set to NULL.

**ON DELETE CASCADE**

A FOREIGN KEY with ON DELETE CASCADE means that if a record in the parent table is deleted , then the corresponding records in the child table will be deleted automatically .

**Not Null constraint with Alter statement:**

ALTER TABLE table2 MODIFY (col1 DATATYPE(size) NOT NULL);

**Remove Constraint:**

ALTER TABLE table1 DROP CONSTRAINT cons\_name;

**Example:**

ALTER TABLE person DROP CONSTRAIN dob\_nn;

UNIQUE:

* Can have max 32 col together
* Not applicable for LONG, LOB, RAW, BFILE etc same as PRIMARY KEY

Unique column level:

CREATE TABLE supplier ( supp\_id NUMBER(4) CONSTRAINT supid\_unq  
UNIQUE, name VARCHAR2(20), contact\_number NUMBER(15));

**Unique with Create statement (Table Level):**

CREATE TABLE supplier( supp\_id NUMBER(4), name VARCHAR2(20), contact\_number NUMBER(15), CONSTRAINT supid\_unq UNIQUE(supp\_id));

**ALTER TABLE table2 ADD CONSRAINT cons\_name UNIQUE(col1);**

**ALTER TABLE location DROP CONSTRAINT newsim\_unq;**

**CHECK column level:**

CREATE TABLE person

(

name **VARCHAR2(10),**

gender **CHAR(1)** CONSTRAINT gndr\_chk CHECK(gender IN ('M','F'),

dob **DATE,**

);

Table level:

CREATE TABLE person

(

name VARCHAR2(10),

gender CHAR(1),

dob DATE,

CONSTRAINT gndr\_chk CHECK(gender IN ('M','F')

);

ALTER TABLE person ADD CONSTRAINT gndr\_chk CHECK(gender IN ('M','F'));

ALTER TABLE person DROP CONSTRAIN gndr\_chk;

Functions:

MATH FUNCTIONS

* ABS
* MOD - MOD(10,4) 🡪 2
* POWER & SQRT
* FLOOR & CEIL
* TRUNC & ROUND

CHARACTER FUNCTIONS:

* INITCAP
* LOWER & UPPER
* CONCAT
* LPAD & RPAD
* TRIM
* SUBSTR
* REPLACE
* LENGTH
* INSTR

DATE FUNCTIONS:

* MONTHS\_BETWEEN()
* ADD\_MONTHS()
* LAST\_DAY()
* NEXT\_DAY()

CONVERSION FUNCTIONS:

* NVL
* NVL2
* NULLIF

AGGREGATE FUNCTIONS:

* SUM
* AVG
* MAX
* MIN
* COUNT

SQL JOINS:

* CROSS JOIN
  + - select \* from tA, tB // cartesian product
* INNER JOIN
  + EQUI JOIN
    - select tA.c1, tA.c2, tB.c1… from tA, tB where tA.c1=tB.c1;
  + NON EQUI JOIN
    - select tA.c1, tA.c2, tB.c1… from tA, tB where tA.c1 BETWEEN tB.c2 and tB.c3;
* OUTER JOIN
  + LEFT OUTER JOIN
    - select tA.c1, tA.c2, tB.c1… from tA, tB where tA.c1=tB.c1(+);
  + RIGHT OUTER JOIN
    - select tA.c1, tA.c2, tB.c1… from tA, tB where tA.c1(+)=tB.c1;
  + FULL OUTER JOIN
    - select tA.c1, tA.c2, tB.c1… from tA, tB where tA.c1=tB.c1(+) UNION select tA.c1, tA.c2, tB.c1… from tA, tB where tA.c1(+)=tB.c1;
* SELF JOIN
  + - Select A.\* FROM tableA A, tableA B WHERE A.col = B.col1 AND condition;

// on same table

**Guidelines For Sub queries :**While defining sub queries , certain guidelines need to be followed.

* A sub query must be enclosed within parenthesis.
* A sub query must appear on the right hand side of the operator.
* A sub query must not contain an ORDER BY clause.

**Sub query types**Sub queries can be classified into below types based upon the logic used.

* Single row sub query
  + returns single row
  + = != < <= > >= are allowed
* Multi row sub query
* Multi column sub query
* Co-related sub query

**Usage of ALL operator**

* ALL → Compares values to every value returned by the sub query.
* < ALL → Less than the minimum value in the list
* > ALL → More than the maximum value in the list

*SELECT \* FROM employee WHERE did =  
           (SELECT did FROM employee WHERE name='RAJU');*

*SELECT \* FROM employee WHERE salary >  
              (SELECT salary FROM employee WHERE name='RAJU');*

*SELECT \* FROM employee WHERE salary = (SELECT MAX(salary) FROM employee);*

 SELECT did,MIN(salary) FROM employee GROUP BY did HAVING MIN(salary) >  
          (SELECT MIN(salary) FROM employee WHERE did=103);

***SELECT e.\* FROM employee e,grade g WHERE did = (SELECT did FROM employee WHERE name='SANTOSH') AND g.grade='B' AND e.salary BETWEEN g.losal AND g.hisal;***

The above Query displays the details of those employees who belongs to the department SANTOSH belongs to and are falling in the salary range of grade 'B'.

***SELECT \* FROM employee WHERE designation IN (SELECT designation from employee e,department d WHERE e.did=d.did AND d.deptname='PRODUCTION');***

The inner query will return the different designated employee details who are working in the PRODUCTION department. Since the inner query is returning more than one record, IN operator is used to link between outer and inner query.Finally the outer query will retrieve all the records whose designations are matching with the designations of PRODUCTION department.

***SELECT \* FROM employee WHERE salary IN (SELECT MAX(salary) FROM employee GROUP BY designation);***

The above query displays the records of all the employees who are earning maximum salary for each designation group.

*SELECT \* FROM employee WHERE salary >ANY (SELECT salary FROM employee WHERE did=102) AND did!=102;*

The above query displays the details of those employees who are **earning more than the minimum salaried employee** belonging to department number 102.

**SELECT \* FROM employee WHERE salary >ALL (SELECT salary FROM employee WHERE designation='SENIOR PROGRAMMER');**

The above query displays the details of those employees who are earning more than the **maximum salaried employee**, who is a SENIOR PROGRAMMER.

***SELECT \* FROM employee WHERE salary <all (SELECT salary FROM employee WHERE designation='SENIOR PROGRAMMER');***

The above query displays the details of those employees who are earning less than the minimum salaried employee, who is a SENIO PROGRAMMER.

**Multi column sub query:**

***SELECT \* FROM employee WHERE (did,salary) IN (SELECT did,salary FROM employee WHERE designation='EXECUTIVE') AND designation!='EXECUTIVE;***

The above query displays the details of those employees who belong to the same department and earning the same salary as of the EXECUTIVE designated employees.

The same logic can even be written in the below format:  
SELECT \* FROM employee WHERE did IN (SELECT did FROM employee WHERE designation='EXECUTIVE') AND salary IN (SELECT salary FROM employee WHERE designation='EXECUTIVE')  
AND designation!='EXECUTIVE;

**Co-related sub query:**

***SELECT \* FROM employee e1 WHERE salary = (SELECT MIN(salary) FROM employee e2 WHERE e1.designation=e2.designation);***

The above query displays the records of all the employees who are earning minimum salary for each designation group.

**Order of Precedence**

* First the Outer query is executed.
* Passes the executed column value to the Inner queries WHERE clause.
* Now the Inner query is executed.
* The result of the Inner query is passed to the Outer queries WHERE clause.
* Depending on the provided value the condition is qualified for the specific record.
* If successful displays the output.

Sub queries in DML:

***UPDATE employee SET did=(SELECT did FROM employee WHERE did='PURCHASE') WHERE did=(SELECT did FROM employee WHERE did='SALES');***  
The above queries updates the department numbers of those employees who  
are belonging to SALES department to PURCHASE department.

***DELETE FROM employee WHERE salary > (SELECT AVG(salary) FROM employee);***  
The above query deletes all the records from the employee table whose salary is more than the average salary of all the employees in the organization.

Sub queries with create and insert:

***CREATE TABLE employee\_bkp AS SELECT name,designation FROM employee WHERE 1=2;***  
The above query creates an empty backup table to store only name and designations.

*INSERT INTO employee\_bkp SELECT name,designation FROM employee WHERE did IN (101,103);*

* Sql rownum starts at 0
* DISTINCT clause eliminates duplicates