A relational database system is a system that allows you to create, update & insert in a relational database

Unit-2

Part 1 : Relational Data Model & Algebra

* **Relational Model** → discovered by Dr. E.F. Codd.

→ "Relational model is today the primary data model for commercial data processing applications".

→ The relational model is an abstract theory of data that is based on the mathematical concept of relations.

→ In relational database, info is stored in tabular form or in tables.

→ The Rel. model is concerned with three aspects of data: data structure, data integrity & data manipulation.
(constraints, accuracy of data)

table → Relation
row → tuple
(column → attribute)

↳ **Structure of Relational Databases / Defination of Relation**

Consider the account table → Relation.

Attribute

account_number	branch_name	balance
A-101	Kanpur	500
A-102	Allahabad	400
A-201	Delhi	900
A-215	Sonepat	700
A-217	Panipat	150
A-222	Karnal	700
A-305	Punjab	350

Tuple ← A-215 row

Domain: 500, 400, 900, 700, 150, 700, 350

Cardinality

→ tuple variable.

Degree

DSDI DM

— Nav

All basic data structures used in Relational database.

1. **Relation** : In Relational Model, name of table is known as relation.
   Ex: account table is known as relation.

2. **Attributes** ~~Tuple~~ : A column in an relation is called ~~tuple~~. Attributes are or the characteristic property of any relation is attribute
   Ex.:
   account_table has three column headers i.e. acc_no, branch_name & balance. we refer to these headers as attributes.

3. **Domain** : It is the set of all permitted values or any info about any attributes called the domain of that attribute.
   Ex:
   As the attribute branch_name, the domain is set of all branch names.
   domain
   let D_1 denotes the set of all acc_no
   " D_2 " " " " " branch_names
   " D_3 " " " " " balances.

4. **Tuple** : A row in an relation is called tuple.
   Ex:
   any row of a account must consists of a 3-tuples (v_1, v_2, v_3) where.
   v_1 is an acc_no. (v_1 is in domain D_1)
   v_2 is an branch_name (v_2 is " " D_2)
   & v_3 is an balance (v_3 " " " D_3).
   i.e.
   A-215 (acc_no), Sonepeet (branch_name), balance (700) represent tuple.

So, account will contain only a subset of the set of all possible rows/tuples account is the set of $D_1 \times D_2 \times D_3$

A table of n-attributes must be a subset of $D_1 \times D_2 \times \cdots \times D_{n-1} \times D_n$.

5. **Tuple variable** : Tuple variable is the part of tuple or row, which is info or data stored in relation of for any attribute.

Ex:

Sonepat is tuple variable is the part of 2nd tuple or row, _____ _____ in relation As branch-name attribute.

6. **Cardinality** : No of tuple in a ~~table~~ relation is called cardinality.

7. **Degree** : No of ~~Column~~ Attributes in a relation is called its Degree.

* ~~Set Operations~~

* **Relational Algebra**

→ It is a procedural query language

→ It specifies the operations to be ~~Perm~~ Performed on existing relations to desire result ~~ees~~ relations

→ A sequence of relational algebric operations forms relational algebric expression.

→ The result of the relational algebric expression is also a relation.

→ The relational algebra is very important due to many reasons such as -

ⓐ Firstly, it provides a basic foundation for relational model operations.
ⓑ Secondly, it is used as basis for implementing & optimizing queries in RDBMS's
ⓒ Thirdly, some of the basic concepts of relational algebra are incorporated into the SQL language.

→ The relational algebra uses various logical connectives [^(and), v(or), ¬(not)] & comparisons operators (<, <=, =, ≠, >=, >) to construct & composite & more complex queries.

→ Operations in Relational Algebra

(A) Set-oriented operations
(B) Relation oriented operations.

(A) Set-Oriented Operations

1. Union operation :→ That union operation is a binary operation that is used to find union of relations.
   → Here relations are considered as sets.
   → So, duplicate values eliminated
   → It is denoted by "∪"

Conditions for union operation:
(i) Both the relations have same no of attributes
(ii) Data type of their corresponding attribute must be same.

Ex: This database contain two Tables Employee & Student. & the relationship is that an employee can also a student & vice-versa.

Ex (Union): Name of all employees & name of all students together then the query is

$\Pi_{Name}(Employee) \cup \Pi_{Name}$

Employee

Eid	Name	Salary
1E	John	61000
2E	Ramesh	5000
3E	Smith	82000
4E	Jack	61000
5E	Nile	151000

Student

Sid	Name	Fees
1S	Smith	1000
2S	Vijay	950
3S	Qaseer	2000
4S	Nile	1500
5S	John	950

(Employee - Student relations)

Ex (Union): name of all employee & name of all students together. Then the query is

$\Pi_{Name}(Employee) \cup \Pi_{Name}(Student)$

So, Employee ∪ Student

Name
John
Ramesh
Jack
Vijay
Qaseer
Smith
Nile

② **Intersection operation**: It is an binary operation that is
→ set intersection is used to find common tuples b/w two relations
→ It is denoted by (∩)
→ Rules of set union are also applicable here.

Ex: name all the employee those are also students then the query is

$$\Pi_{Name}(Employee) \cap \Pi_{Name}(Student)$$

Name
John
Smith
Nik.

③ **Set-difference operation**.
→ It is an binary operation which is used to find tuples that are present in one relation but not in other relation.
→ It is denoted by " − ".
→ It removes the common tuples of two relations.

Ex: name of those employees that are not students then the query is

$$\Pi_{Name}(Employee) - \Pi_{Name}(Student)$$

Name
Ramesh
Jack

④ **Cartesian Product operation**:
→ It is a binary operation which is used to combine information of any two relations.

→ Suppose a relation R_1 is having m tuples & other relation R_2 is having n tuples then $R_1 \times R_2$ has m*n tuples.

→ It is denoted by (x)

→ It is also known as Cross-join

Ex: Cartesian product of relation R & S is

Query use →

R

	A	B	C
m	1	1	2
	2	3	3
	3	1	3

(x)

S

C	D	
3	4	n
4	5	

Query is : R x S.

	A	B	C	C	D
m x n	1	1	2	3	4
	1	1	2	4	5
	2	3	3	3	4
	2	3	3	4	5
	3	1	3	3	4
	3	1	3	4	5

(x+y)

Where
x+y = Column headers
m x n = domain

(B) **Relational Oriented Operation:**

1. **Selection or Restriction operation (σ)** or horizontal subset

→ It is used to extracts tuples (row) from a relation depending upon given ~~relation~~ condition

→ It is an unary operation.

→ It is denoted by (σ) "Sigma"

R

A	B	C
1	1	2
2	2	1
3	1	2

$\sigma(R)$ "R.B < R.C

A	B	C
1	1	2
3	1	2

$\pi(R)$ BC

B	C
1	2
2	1

discard the (1 2) same value

→ Ex: employees having salary more than 9,000 from relation Employee, the query is

$$\sigma_{salary > 9,000}(Employee)$$

Result is

Eid	Name	Salary
1E	John	101000
5E	Nile	151000

Note:

If u want name of all employees having salary <7000 then the query is

$$\sigma_{salary<7000}[\pi_{Name}(Employee)]$$

for condition / tuples / rows — for name / column / attributes

2. Projection Operation:

→ It is used to extracts column (attributes) as vertical subset from a given relation.
→ It is an unary operation.
→ It applies only on a single relation (table) at a time.
→ It is denoted by (π) "pi"
→ Ex: name of all the employees & their salary from employee relation then query is:

$$\pi_{Name, Salary}(Employee)$$

Name	Salary
John	10,000
Ramesh	51000
Smith	81000
Jack	61000
Nile	151000

③ **Division operation:**

→ It is useful in special kind of queries that include the phrase "for all".

→ It is denoted by (÷)

→ It is like the inverse of cartesian product.

→ Ex:

A

X	Y
x_1	y_1
x_1	y_3
x_1	y_2
x_4	y_1
x_5	y_5
x_2	y_3
x_3	y_4
x_4	y_1

B1

Y
y_1
y_3

B2

Y
y_1
y_5

B3

Y
y_5
y_2
y_4

Query A ÷ B1 gives:

X
x_1
x_4
x_2

A ÷ B2 gives:

X
x_1
x_4
x_5

A ÷ B3 gives:

X
x_5
x_1
x_3

(Result of division operation)

④ **Join operation:**

→ ~~The join operation denoted by ⋈.~~

→ It is used to combine related tuples from two relations into single tuples.

→ This operation is very ~~useful~~ important for any relational database with more than a single

relation bec. it allows us to process relationship among relations.

→ Join operation can be done by using ($\times$, π, σ)

→ Syntax:

$R(A_1 \dots A_n) \bowtie_{\langle condition \rangle} S(B_1 \dots B_n) = Q(A_1 \dots A_n, B_1 \dots B_n)$

→ It is of two types:

ⓐ Natural join: ⓑ outer join

ⓐ **Natural Join:**

→ It is used to join two relations having any no of attributes.

→ It is denoted by ~~symbol~~ ($\bowtie$)

→ It also optimize the query as Cartesian Product gives unnecessary results & set-union & set-intersection operations are applicable only on those relations that have equal no of attribute with same data type.

→ Ex: ~~name of all emp~~ Consider the relations employee & Department

Employee

Eid	Name	Salary	Dept_id
1	A	5,000	10
2	B	8,000	20
3	C	2,000	20
4	D	10,000	10

Department

Dept_id	Dept_Name
10	Sales
20	Purchase

~~Query~~ name of all employees from relation Employee with their respective department names.

Query is :

$$\Pi_{Name,\ Dept\text{-}Name}\left[\begin{array}{l} Employee \bowtie Department \\ Employee.Eid = Department.Dept\_id \end{array}\right]$$

Result is

Name	Dept-Name
A	Sales
B	Purchase
C	Purchase.
D	Sales

(5) Outer join :

→ It is an extension of natural join operations.

→ It deals with the missing information caused by natural join operation.

→ Suppose if we need all info. of all employee & all Student in a single relation that is obtained by outer join.

→ Ex : let us consider two relations. Employee & Student

Employee.

Eid	Name	Salary
1E	John	10,000
2E	Ramesh	51000
3E	Smith	87000
4E	Jack	61000
5E	Nile	15000

Student

Sid	S-name	fees
1S	Smith	1000
2S	Vijay	950
3S	Gaurav	2000
4S	Nile	1500
5S	John	950

Natural Join

(Employee ⋈ Student) gives the result as

Eid	Sid	Name	Salary	fees
1E	5s	John	10,000	950
3E	1s	Smith	8,000	1000
5E	4s.	Nile	15,000	1500

In this result, info about Ramesh, Jack & vijay are missing. So, outer join is used to avoid these loss of information.

→ There are three types of outer joins.

(i) Left outer join:

* It is used to take all tuples of relation that are on the left side whether they are matching with tuples of right side or not.
* It is denoted by (⟕)
* Ex: (Employee ⟕ Student) gives

Eid	Sid	Name	Salary	fees
1E	5S	John	10,000	950
2E	NULL	Ramesh	5,000	NULL
3E	1S	Smith	8,000	1000
4E	NULL	Jack	6,000	NULL
5E	4S	Nile	15000	1500

(Result of Left outer join)

(ii) Right outer join:

* It is used to take all tuples of relation that are on the right side whether they are matching with tuples of left side or not.
* It is denoted by (⟖)

* Ex: (Employee ⟖ Student) gives

Eid	Sid	Name	Salary	Fees
3E	1S	Smith	8000	1000
NULL	2S	Vijay	NULL	950
NULL	3S	Gaurav	NULL	2000
5E	4S	Nile	15000	1500
1E	5S	John	10,000	950

(Result of right outer join)

(iii) **Full outer join:**

* It is used to take all tuples from left & right relation whether they match with each other or not.
* It is denoted by (⟗)
* Ex: (Employee ⟗ Student) gives

Eid	Sid	Name	Salary	Fees
1E	5S	John	10000	950
2E	NULL	Ramesh	5000	NULL
3E	1S	Smith	8000	1000
4E	NULL	Jack	6000	NULL
5E	4S	Nile	15000	1500
NULL	2S	Vijay	NULL	950
NULL	3S	Gaurav	NULL	2000

(Result of full outer join)

* **Relational Calculus**

↳ An alternative to relational algebra is relational calculus.

→ It is a query system where queries are expressed as variables & formulas on these variables.

→ It is a non-procedural or declarative by nature.

→ In this query represents only results & hides the procedure to find the results.

→ Relational calculus has two variables : ~~the first~~ (i.e. ~~the~~)

(a) (1) ~~the~~ Tuple Relational Calculus.

* In this every variables takes on tuples as values.
* Every query in tuple relational calculus is expressed by a tuple calculus expression which is the basic construct i.e

$$\{ \frac{T}{F(T)} \}$$

Here, T is the set of tuple variables & F is the formula involving T

Ex:

* Ex: Consider the relation Employee & Department.

Employee

Eid	Name	Salary	Dept-id
1	A	9000	1
2	B	8000	2
3	C	7000	1
4	D	6000	2
5	E	5000	3

Department

Dept-id	Dept-name
1	Becca IT
2	CSE
3	BCA

(tuples higher)

queries: find all employees having salary more than 7,000.

query: $\{t \mid t \in Employee \wedge t[salary] > 7000\}$

$t \in r$ $\quad$ $t[A]$ Qualified

$\{t \mid P(t)\}$ $\quad$ $\{t \mid t \in r \wedge t[s] > y\}$

It means we find those tuple (t) that belongs to relation (r) Employee & having salary more than 7000. The result is

Eid	Name	Salary	Dept-id
1	A	9000	1
2	B	8000	2

(Employees having salary > 7000)

(2) Domain Relational Calculus

→ In Domain R.C. every variables range over the underlying domains.

→ Every query in D.R.C. is expressed by a Domain Calculus expression which is the basic construct i.e.

$[D|F(D)]$ or $\{(d_1, d_2 \cdots d_n) | F(d_1, d_2, \cdots d_n)\}$

Here, D is the set of domain variables $(d_1, d_2, \cdots d_n)$ & F is the formulae involving $D(d_1, d_2, \cdots d_n)$

→ The result of this query is set of all tuples for which the formula is true.

→ Ex: Again considering same relation employee & department(d)

Query: Find all employees having salary more than 7000.

query: $|<E, N, S, D| <E, N, S, D> \in Employee \wedge S > 7000|$

where E = Emp-id
N = Name
S = Salary
D = Dept-id

result is

Eid	Name	Salary	Dept-id
1	A	9000	1
2	B	8000	2

Unit-II

Integrity Constraints & Intro. to SQL :-

Integrity Constraints :

→ These are the rules or constraints to the database to keep data stable, accurate or consistent.

→ To keep database consistent we have to follow some rules known as integrity rules or integrity constraints. such as

→ Also, Integrity constraints ensure that changes made to the database by authorized users do not result in a loss of data consistency.

1) Domain Constraints :

* The restrictions which we applied on domain are known as domain constraints.
* These restrictions are applied to every value of attribute.
* These restrictions include : data types
  (i) data types (integer, varchar, char, float, time)
  (ii) length or size of variable
  (iii) ~~Checks (like value NOT NULL etc)~~ etc.
  (iv) is null value allowed etc

* Eg: Create table employee
(

Eid	integer (2),
Name	Char (20),
Age	integer (2),
salary	integer,

);

Employee

Eid	Name	Age	Salary
1	A	22	5,000
2	B	20	6,000
3	C	19	ABC → Not allowed bec. Salary has integer data type.
4	D	18	101000

(Domain constraint on relation employee)

2) **Key constraints**

~~In any relation R, if attribute A is primary key~~

* In any relation R, if any attribute acts as a Key attribute (primary / super / candidate key) then it must have unique value.
* Duplicates values in key attribute are invalid
* Eg: Create table employee
  (Eid integer(2) PRIMARY KEY,
  Name char(20),
  Age integer(2),
  Salary integer(5),
  );

Employee

Eid	Name	Age	Salary
1	A	22	5000
2	B	20	6,000
2 ← Invalid duplicate value	C	19	8,000
3	D	18	10,000

(Key constraints on relation Employee)

3) Check Constraints :

* The check constraints is used to ensure that attribute value satisfy specified conditions.

* Eg : Salary int(5) check.(Salary > 5000)

Employee

Eid	Name	Age	Salary
1	A	19	3000 → invalid
2	B	20	201000
3	C	21	361000
4	D	22	401000

* The check constraint is used to limit the value range that can be placed on a column.

4) NOT NULL Constraints

* The NOT NULL Constraints is ensure that attribute value does not have NULL value.

* Eg : Name char(20) NOT NULL;

Employee

Eid	Name	Age	Salary
1	A	19	30,000
2	B	20	20,1000
3		21	40,000
4	D	22	50,000

→ invalid.

5) **Entity - Integrity Constraints:**

* The entity integrity constraints states that primary key can not be NULL.
* There must be a proper value in the primary key field.
* This is bec. the primary key value is used to identify individual rows in a table.
* If there were NULL values in the primary key it would mean that we could not identify those rules.
* Eg: Eid integer(2) PRIMARY KEY

Employee

Eid	Name	Age	Salary
1	A	19	3,000
2	B	20	4,000
☐ ← This is invalid Eid is primary key.	C	21	5,000
4	D	22	6,000

6) **Tuple - Uniqueness Constraints:**

* In any relation R, all tuples in relation R must have distinct values.
* In other words, duplicate tuples within a single relation is not allowed.
* Ex:

Eid	Name	Age	Salary
1	A	19	3,000
2	B	20	4,000
3	C	21	5,000
2 ← Not allowed (duplicate tuple)	A	19	3,000

7) **Referential Integrity Rule.**

* The referential integrity constraints specified b/w two table & it is used to maintain the consistency among rows b/w two tables.

* Cor. Ex: Create table Employee (
Eid Integer(2)
Name Char(20),
Age Integer(3),
Salary Integer(5),
Emp-D-id varchar(3) FOREIGN KEY
REFERENCES Department (D_id)
);

Employee (Referencing Relation)

Eid	Name	Age	Salary	Emp-D-id
1	A	19	31000	D_1
2	B	20	41000	D_1
3	C	21	51000	D_2
4	D	22	61000	D_3
5	E	23	71000	D_2

Department (Referenced Relation)

Did	D-name
D_1	M tech
D_2	B tech
D_3	BBA

* The relation where common attribute act as a foreign key is known as (Referencing Relation.)

* And the relation where it act as primary key is known as (Referenced Relation)

* Limitations:

(i) You can not delete a record from a referenced relation if the

matching record exist in referencing relation.

(ii) You can not change a Primary Key value in the referenced table if that record has related record in a referencing relation.

(iii) You can not enter a value in the foreign key field of referencing table that does not exist in the primary key of referenced relation.

(iv) You can not put a NULL value in a foreign key.

Basic Structure & concept of DDL, DML, DCL:

⇒ SQL developed by Chamberlin & Raymond Boyce in mid-1970s

* SQL is a non-procedural language used for quering upon relational database.
* SQL consists of a set of facilities for defining, accessing & managing relational databases.
* All tasks related to relational data management - creating tables, quering the database, deleting etc, can be done by using SQL.
* SQL statements can be invoked either interactively in a terminal session or by embedding them in application programs.

↳ Characteristics:

1. SQL is extremely flexible.
2. It is a free formated language.
3. It has relatively few commands
4. It is a non-procedural language.

@Kash

↳ Advantages:

1. SQL is a high level language that provides greater degree of abstraction than procedural languages.
2. SQL is a unified language. The same lang. can be used to define data structures, querying data, insert, modify, delete data & so on.
3. The language is simple & easy to learn. It can handle complex situations very efficiently.
4. SQL processes set of records rather than just one record at a time.

↳ ~~Parts (components) of SQL language~~

↳ Parts (components) of SQL language:

SQL is divided into 3 major parts:

(1) Data-Definition Language (DDL)

* It is used to specify a database conceptual schema using set of definations
* It supports the definition or declaration of database objects.
* The DDL is also used to specify additional properties of data.
* SQL commands which comes under DDL are

(i) CREATE Table.
(ii) Alter Table
(iii) Rename Table
(iv) Truncate Table
(v) Drop Table

(i) **Create Table Command:**

* The "create table" command defines each column of the table uniquely.
* Each column has min. of three attributes i.e. a name, datatype & size
* Each column defination is a single clause in the create table syntax.
* Each ~~defination~~ table column defination is separated from by a comma.
* Finally, the SQL statement is terminated with a semicolon(;)

Syntax

Create Table <Table Name> (<column name 1><Data type>(<sizes>),
<column_name 2><Data type>(<sizes>);

(ii) **Alter Table command:**

* The structure of a table can be modified by using ALTER TABLE command.
* Alter table allows changing the structure of an existing table.
* With Alter Table, it is possible to add or delete columns, create or destroy indexes, change the data type of existing columns or rename columns or table itself.

* Alter Table works by making a temporary copy of the original table. The alteration is performed on the copy, then the original table is deleted & the new one is renamed.

Syntax:

(a) Adding a new column:

alter table <table name> add (<new-column-name-1><data type>(<size>), <new-column-name-2><data type>(<size>));

(b) Dropping a column

alter table <table_name> drop column <column-name>;

(c) Modifying Existing ~~column~~ Table:

alter table <table-name> modify (<column-name><new-data_type>(<new-size>));

(iii) **Renaming Table**: This command is used to rename an existing table.

Syntax

rename <table-name> to <new-table-name>;

(iv) **Truncating Tables**:

* Truncate table empties a table completely.
* Logically, this is equivalent to DELETE Command that delete all rows
* Truncate operations drop & re-create the table, which is much faster than deleting rows one by one.
* The no of deleted rows are not returned.

Syntax: truncate table <table-name>;

(v) **Drop table command**:

* Sometimes tables within database become obsolete & need to be discarded.
* In such situations using the DROP TABLE statement with the table name can destroy a specific table.
* If table is dropped, all records held within it are lost & cannot be recovered.

Syntax

drop table <table_name>;

(2) Data Manipulation Language (DML)

* DML is a language that enables users to access or manipulate data as organized by the appropriate data model.
* ~~The type of access are~~ SQL commands which are similar:
  (i) Insert into command
  (ii) Select command
  (iii) Update command
  (iv) Delete command.
* There are basically two types of DML -

(a) Procedural DML - require a user to specify what data are needed & how to get those data.

(b) Declarative (or non-procedural DML) - require a user to specify what data are needed without specifying how to get those data.

(i) Insert into Command

* Initially, a newly created relation is empty.
* We can use the insert command to load data into the relation.

Syntax

insert into <table_name> values (<expression 1>, <expression 2>);

OR

insert into <table_name> (<column 1>, <column 2>) values (<exp 1>, <exp 2>);

(ii) Select Command

* The SQL select statement returns a result set of records from one or more tables.

* A Select Statement retrieves zero or more rows from one or more database tables or database views.

Syntax

(a) to retrive all attribute :

Select * from <table-name> ;

(b) to retrieve selected attribute :

Select <column 1> <column 2> from <table name> ;

(iii) Update Command :

* The SQL update command is used to update existing records in the tables
* An UPDATE command can change the data of one or more records in a table.

Syntax

Update <table name> set <column 1> = <new value>
where <condition>

(iv) Delete Command :

* A SQL Delete command is used to delete data from a table.
* With Where Clause we can delete particular tuple from a table.
* And without Where Clause all tuples are deleted from the table.

Syntax : delete from <table_name> where <condition>

(3) Data Control Language (DCL)

* DCL is a language that provides commands that help the DBA to control the & database.

* DCL defines two commands:

(i) Grant: → used to provide ~~gives user access~~ privileges on the database objects to the users.

Syntax

(a) to allow a user to create session

grant create session to username;

(b) to allow a users to create table.

grant create table to username;

~~(c) to provide user with some space on tablespace to store Table.~~

(c) to grant all privilege to a user:

grant sysdba to username;

(d) to grant permission to create any table.

grant create any table to username;

(e) to grant permission to drop any table.

grant drop any table to username;

(ii) Revoke: → It removes user rights or privileges to ~~Take back permission from user~~ database objects.

Syntax

(a) to take back permissions

revoke create table from username

* DCL Defines two commands

(i) Commit Command - It is the transactional command used to save changes invoked by a transaction to the database.

Syntax:

Commit [work];

Where Commit is the keyword & WORK is the optional keyword & it is used to make the command more user friendly.

Ex: DELETE from Emp
Where Basic > 7000;
Commit ;

After deletion, a COMMIT comm. is issued to save changes to the database, Completing the transaction.

(ii) Rollback Command - It is used to undo transactions that have not already been saved to the database.

Syntax:

Rollback [work];

Ex:

DELETE from Emp
Where Basic < 4000
Rollback;

Here, first statement (Delete) will delete the records of all the employees whose (basic < 4000)

But the 2nd statement (Rollback) removes the effect of 1st statement hence, there is no change in the Employee table.

* DCL defines two commands:

(i) GRANT Statement: It is used to provide access or privileges on the database objects to the users.

* Syntax: In a multi-user database management system, it is required to grant diff permissions for security purposes.

Syntax:

```
GRANT <privilege-list> | ALL
    ON <object>
    TO <user-list> | PUBLIC
    [With Grant option]
```

Where,

- privilege list - Specifies the permission to be granted to users like ALTER, DELETE, UPDATE, CREATE etc.
- ALL - specifies all the permissions.
- Object - specifies name of tables or their columns
- User-list - specifies name of user to which permissions are granted.
- Public - specifies all users
- [With grant option] - specifies that the user in user-list can give permission to any other user that were granted to him. [if specified]

Eg: Grant all the permission on table Emp to all users:

```
GRANT ALL
ON Emp
TO PUBLIC;
```

(ii) **Revoke Statement** : It is used to take away any authority from a user that was granted earlier.

Syntax :
```
REVOKE  <privilege-list> | ALL
   ON      <table name> [(column-comma-list)]
  FROM     <user-list> | PUBLIC.
```

Eg : Revoke the update permission on table Dept - from Rohan

```
REVOKE Update
   ON     Dept
  FROM    Rohan;
```

Aggregate Functions :

* Agg. functions are functions that takes a collection of values as input & return a single value.
* It can be applied to all rows in a table or to a subset of the table specified by a where clause.
* SQL offers Six built in aggregate functions :

Syntax : select Aggregate-function(column-name) from tablename.

(i) **sum** : returns the sum of the numeric values in a given column.

Eg :
to calculate the total salary for employees of grade 'A', i.e :

```
select sum(salary)
  from employee where grade = 'A';
```

(ii) **AVG** : returns the avg of the numeric values in a given column.

Eg :
to calculate the avg. salary for employees of grade 'A'

```
select avg(gross)
  from employee where grade = 'A';
```

Eid	Name	Salary	Grade	City
1	Arun	10,000	A	Delhi
2	Ram	20,000	B	Haryana
3	Anil	30,000	A	Delhi

(VII) First() : It returns 1st value of a specified column.
select first(salary) from employee

(VIII) Last() : It returns last value of a specified column.
select last(salary) from employee

(III) **max** : returns the maximum of numeric values in a given column.

eg :
give the max. salary for employees i.e.
select max(salary)
from employee;

(IV) **min** : returns the minimum of numeric values in a given column.

eg :
give the min salary for employees i.e.
select min(salary)
from employee;

(V) **Count(*)** : returns the ~~total~~ no of rows in a table.
~~given column.~~

eg : to count the no of employees in employee table i.e.
select ~~count(DISTINCT~~ count(*)
from employee;

(VI) ~~count(*)~~ : **Count(col)** : returns the total no of values in a given column.

eg : to count no of cities the club employees belong to i.e.
Select count (DISTINCT city)
from employee;

NULL values :

* SQL allows the NULL is the term used to represent a missing value.
* A NULL value in a table is a value in a field that appears to be blank.

* By default, a table colum can hold NULL values.
* A field with null value is a field with no value.
* Syntax — basic syntax while creating table

```
Create table Employee( P_id int NOT NULL,
                       Name NOT NULL,
                       Address int,
                       City NOT NULL,
                       Primary Key ( P_id )
                     );
```

- Here, NOT NULL signifies that column should always accept an explicit value of the given data type.
- In these are one column where we did not use NOT NULL, which means this column could be NULL.

* Example — In order to check for a NULL value, we must use the "IS NULL" OR "IS NOT NULL" operators

Consider the full table, Employee having the full records :

P_id	Name	Address	City
1	A		Delhi
2	B	Delhi B-16	Chennai
3	C		Mumbai

→ Usage of "IS NULL" operator : select P_id, name, Address from Employee where Address IS NULL.

The result will look like

P_id	Name	Address
1	A	
3	C	

→ Ex: Usage of IS NOT NULL:

```
Select P-id, name, address from Persons
    Where Address IS NOT NULL;
```

The result will look like -

P-id	Name	Address
2	B	B-16

Views:

* A view is nothing more than a SQL statement that is stored in the database with an associated name.
* A view is actually a composition of table in the form of predefined query.
* A view can contain all rows from a table or select rows from a table.
* A view can be created from one or many tables which depends on the written SQL query to create a view.
* View which are kind of virtual tables, allows users to do the following:
  - Structure data in a way that users or classes of users find natural or intuitive.
  - Restricted access to the data such that a user can see & sometimes modify exactly what they need & no more
  - Summarize data from various tables which can be used to generate reports.

Need of view - (i) Data security (ii) Data Redundancy.

* **Creating Views** : Database views are created using the CREATE VIEW statement.
Views can be created from a single table, multiple tables, or another view.

Syntax :

```
CREATE VIEW view_name AS SELECT
    Column 1, Column 2, ....
    From table_name
    Where [Condition]
```

Example Consider the Customer table having following records :

ID	Name	Age	Address	Salary
1	A	21	Delhi	40000
2	B	22	Chennai	201000
3	C	23	Bombay	301000
4	D	24	MP	52000
5	E	25	UP	41000
6	F	26	Bihar	31000
7	G	27	Kota	21000

Now, to create a view from customers table. Name customer name, & age.

Q.

SQL> Create view Customers_view As Select name, age.
from CUSTOMERS;

SQL> select * from Customers_view.

this would produce full result.

Name	Age
A	21
B	22
C	23
D	24
E	25
F	26
G	27

* The WITH CHECK Option:

- The with check option is a create view statement option.
- The purpose of the WITH CHECK option is to ensure that all UPDATE & INSERT a satisfy the conditions in the view definition.
- If they do not satisfy the conditions, the UPDATE or INSERT returns an error.
- The foll. is a exmp. of creating view Customers-view with check option:

```
Create view Customers-view As select Name, Age
from Customers where Age IS NOT NULL
with CHECK OPTION;
```

The with check in this case should deny the entry of any NULL views in the new Age column.

* Updating a view: A view can be updated under certain conditions:

(i) The select clause may not contain the keyword DISTINCT
(ii) The select clause may not contain summary functions
(iii) The select clause may " " set "
(iv) " " " " " set operations
(v) " " " " " order by clause
(vi) The from clause " " " " multiple tables
(vii) The where " " " " " subqueries

Ex: To update the age of Rhea A

```
Update Customer-view
Set Age=35
where Name='A';
```

* Inserting Rows into a ~~Table~~ View :

Rows of data can be inserted into a view.
The same rules that apply to the Update command also apply to the Insert command

* Deleting Rows into a view : Rows of a data can be deleted from a view the same rules that apply to the update & insert commands apply to delete command.

Syntax

Delete From <View-name>
Where [condition];

Ex:
Delete from employee view
where Age = 22

* Dropping View : is used to remove a new from a database.

Syntax : Drop View-name ;

Solve Problems

Prob 1: Consider the following relational database.

- Employee (employee-name, street, city)
- works (employee-name, company-name, salary)
- Company (company-name, city)
- Manages (employee-name, manager-name)

(i) find the company that has most employee
(ii) find all the employees in the database who live in the same city as the company they work for
(iii) find the name of all the employees who work for the First corporation bank
(iv) find all the employees who don't work for first corporation bank.
(v) find all those employees who work for first corporation bank & earn more than Rs. 10,000

find the abv. queries give an exp. in SQL

Sol

(i)
```
Select company_name
    from works
Group By company_name
Having Count (Distinct employee_name) >=
        (Select Count (Distinct employee_name)
            from works
            Group By company_name);
```

(ii)
```
select e.employee_name
    from employee e, work w, company c
where e.employee_name = w.employee_name AND e.city = w.city
```

(iii)
```
Select e.employee_name from work w, employee e
    where company_name = 'First Corporate bank';
```

~~Unit-III~~

(iv) Select employee-name form works
where company-name <> 'First corporation Bank';

(v) select e.employee-name, form works e, works w
where w.company-name = 'First Corporation Bank' AND
w.Salary > 10,000;

Prob2: Consider the relation.

Project (P-no, P-name, Chief-Architect)
Employee (E-no, E-name)
Assigned to (P-no, E-no);

(i) Obtain details of employee working on the project named SYSTEM.
(ii) Get details of employee working on both projects P200 & P300.
(iii) Find the name of employee who work on all projects.

And the abv queries give an exp in SQL

Sol
(i) Select e.employee
form employee e, Project P, Assigned-to a
Where ((e.E-no = a.E-no) AND (p.P-no = a.P-no)
AND (P.P-name = 'SYSTEM'));

(ii) Select e.employee
form employee e, project P, Assigned-to a
Where ((e.E-no = a.E-no) AND (p.P-name = 'P200' AND P.P-name = 'P300'))

(iii) Select Distinct e.Eno
form employee e.

Joins

Join clause is used to combine rows from two or more table based on related col. b/w them

Orders table

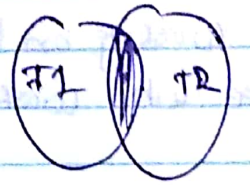
Order id	Cust id	Ord. Date
308	2	9
309	3	7
310	4	8

Customer table

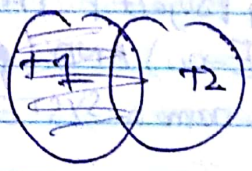
Cust id	Cust-Name	Reference No
1	A	5
2	B	6
5	C	7

Types

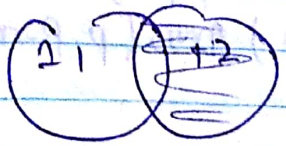
(1) Inner Join - Return records that have matching value in both table.



(2) Left outer Join - Return all records from Left table & then matched records from the R. table



(3) Right outer Join



(4) Full outer Join : Return all records when there is a match in either Left or right table

(1) Inner Join

```
Select col-name
        from table 1
    INNER JOIN table 2
        on
    table 1 . col-name = table 2 . col-name
```

Eg: Select Orders . OrderID , Customers . CustomerName
from Orders Inner Join Customers on
Orders . CustID = Cust . CustID .

(2) Select colname from table 1
Left Join table 2
on tab1 . col-name = tab2 . col-name

(3) ||

(4) Full outer Join -

```
Select col-name
    from table 1
    full outer Join table 2 on
    tab1 . col-name = tab2 . col-name .
```