# UNIT - II

## INTRODUCTION TO RELATIONAL MODEL

Integrity Constraints over relations

enforcing integrity constraints

Querying relational data

Logical database design
- ER to relational
- entity sets to tables
- Relationship sets to tables
- Translating relationship with key cons.
- Translating weak entity, participation const. Hierarchies

Introduction to views, destroying/altering tables & views

Relational algebra

Tuple relational calculus

Domain relational calculus

UNIT-II

## Introduction to Relational Model:-

Relational model represents data in the form of relations or tables. It's a primary data model which is used widely around world. Relational model was proposed by E.F. Codd. Relational model represents how data is stored in Relational Databases.

Consider a relation STUDENT with attributes ROLL_NO, NAME, PHONE

STUDENT

| ROLL_NO | NAME | ADDRESS | PHONE | AGE |
|---|---|---|---|---|
| 1 | RAM | DELHI | 04051 | 18 |
| 2 | RAMESH | ROHTAK | 12345 | 20 |

Important Terminologies:-

1. Attribute
2. Relation schema.
3. Tuple
4. Relation Instance
5. Degree
6. Cardinality
7. Column

Attribute:- attributes are the properties that define a relation.

eg:- Roll_NO, NAME

Relation schema:- It represents name of the relation with its attributes.

eg:- STUDENT (ROLL_NO, NAME .....) is relation schema for student

* If a schema has more than one relation, it is called relational schema.

Tuple:- Each row in the relation is known as tuple.

Eg:-

| | | | | |
|---|---|---|---|---|
| 1 | RAM | Delhi | 9455123451... | 19 |

Relation Instance:

The set of tuples of a relation at a particular instance of time is called as relation instance of STUDENT at a particular time.

Degree:- The no. of attributes in the relation is known as degree.

* Domain:- The set of allowed values for each attribute is called the domain of attributes.

- Roll #: Alphanumeric string
- Firstname, last name: Alpha string
- DOB: date
  ↳ Atomic value (fixed)
- Aadhar: 12 digit number
- Department: Alphastring.
  ↳ Attribute ↓ Domain.

## Relational schema and Instance:

$A_1, A_2, \ldots, A_n$ are attributes

$R = (A_1, A_2, \ldots A_n)$ is a relation schema.

R → schema.

$r \subseteq D_1 \times D_2 \times \ldots D_n$.

Mathematical notation.

Eg:- Instructor = (ID, name, dept_name, Salary)

The collection of data at a particular time is instance

↓ in records

↓ table.

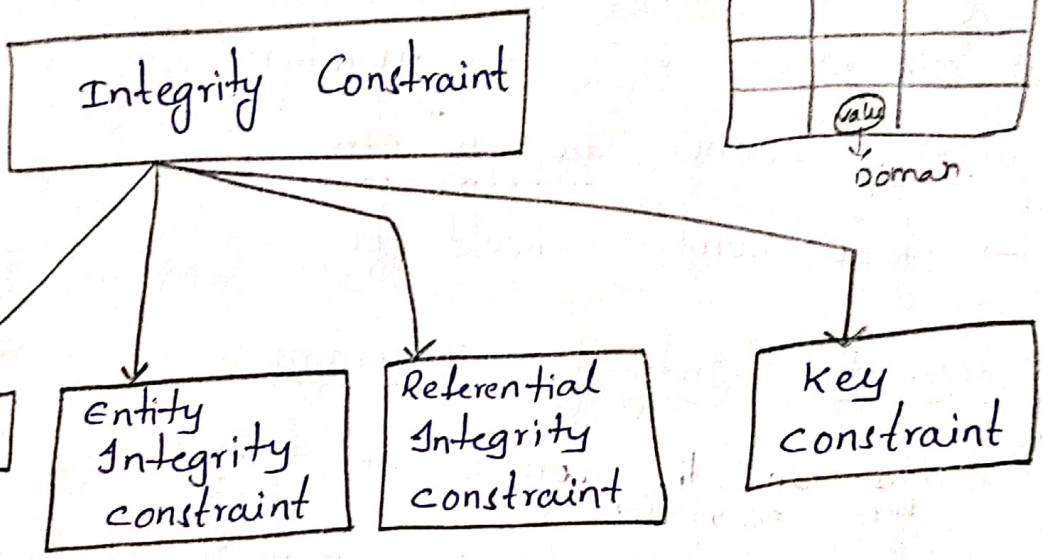
# Integrity Constraints over relations.

Data Integrity refers to accuracy

↓ accuracy, consistency

↓ condition → Important functionality of DBMS

- Integrity Constraints are a set of rules.
- Integrity constraints ensure that data insertion, updating and other processes have to be performed in such a way that data integrity is not affected.
- It guards against accidental damage to database.




Eg:




- key → attribute

* **Key Constraint:-**

In a relation with a key attribute no 2 tuples (rows) can have identical values for key attributes.

- unique in table.

| Id | Name | address |
|---|---|---|
| 1 | | |
| 1 | | |

Relation invalid bcz ID is 1 for both tuple

key attributes

| ID | Name | address |
|---|---|---|
| 1 | bhanu | |
| 2 | bhanu | |

valid relation.

- key attribute cannot have null values.
- If there are more than one such minimal subsets these are called candidate keys.

* Domain Constraint:-

Every attribute is bound to have a specific range of values. An attribute has a set of possible values.

Eg:- *name should be character
→ id " " numbers.
→ Age cannot be negative
→ phone numbers should be 0-9.

* Referential Integrity Constraints:-

→ works on the concept of foreign key.
→ A foreign key is a key attribute of a relation that can be refered in other relation.

Set of rules consists of

| . Insert rule | . update | . delete |
|---|---|---|
| columns that | could attempt to change values of | removes a record from parent table though there is a primary key for child tables. |

## Key Constraints:-

A DBMS key is an attribute or set of an attribute which helps you to identify a row in a relation. They allow you to find the relation between two tables. Keys helps you to uniquely identify the row in a table.

Why?

→ Keys help you to identify any row of data in table. In real world application we have thousands of records moreover the records could be duplicated.

→ Allows you to establish a relationship between and identify the ~~rela~~ relation between tables.

Various keys in DBMS:-

1. Super key
2. Primary key
3. Candidate key
4. Alternate key
5. Foreign key
6. Compound key
7. Composite key
8. Surrogate key.

CAFPS CCS

Super key:- A super key is a group of single or multiple keys which identifies rows in a table. A super key may have additional attributes that are not needed for unique identification.

Eg:-

| Emp S.NO | Emp id | Empname |
|---|---|---|
| 1 | B0 | James |
| 2 | B3 | Roslyn |
| 3 | B5 | Johny |

In the above example emp id and empname are super keys.

Primary key:-

A column or group of columns in a table which helps us to uniquely identify every row in the table is called primary key. In DBMS the same value cannot appear more than once in a table.

Rules:-

- Two rows can't have the same primary key value.
- Every row must have a primary key.
- Primary key field cannot be null.
- The value in a primary key column can never be modified or updated if any foreign key refers.

Eg:-

| Stuid | Name | email |
|---|---|---|
| 11 | Tom | abc@gmail.com |
| 12 | Nick | xyz@gmail.com |

stuid is a primary key.

Alternate key:-

All the keys which are not primary key are called alternate key.

| stuid | Name | email |
|---|---|---|
| 11 | TOM | abc@a. |
| 12 | Nick | xyz@ |

stuid is a primary key name, email becomes the alternate key.

Candidate key:-

A super key with no repeated attribute is called candidate key.

The primary key should be selected from candidate keys. Every table must have atleast single candidate key.

Properties of Candidate key:-

- It must contain unique values.
- Candidate key may have multiple attributes
- Must not contain null values
- uniquely identify each record in table.

| Stuid | Name | email |
|---|---|---|
| 11 | Tom | abc@gmail |
| 12 | Nick | xyz@gmail |
| 13 | Dana | mno@gmail |

stuid, name and email are candidate keys which help us to uniquely identify the student record in table.

Foreign key:-

It is a column which is added to create a relationship with another table. Foreign keys helps us to maintain data integrity and also allows navigation between two different instances of an entity. Every relationship in the model needs to be supported by a foreign key.

Foreign keys are the columns of table that points to the primary key of another table. They act as cross reference between tables.

Eg:- Table may have two or more foreign keys.

Students course table

| course id | course name |
|---|---|
| A004 | Accounts |
| A005 | computers |

↓ Parent table
Primary key

Foreign key

| Std id | Name | courseid |
|---|---|---|
| 1 | sai | A004 |
| 2 | xyz | A005 |
| 3 | abc | |
| 4 | Ram | |

child table

→ Foreign key establishes relationship between two tables.

→ Foreign key is a field in one table that refers to the Primary key in another table.

Example NO :- (2)

Persons table ⇒ Parent table (or) Reference table

orders table ⇒ child table

| Person ID | Name | age |
|---|---|---|
| 1 | Hari | 25 |
| 2 | Sai | 28 |
| 3 | Ram | 30 |

| orders ID | Order no. | Person ID |
|---|---|---|
| 1 | 3246 | 3 |
| 2 | 1234 | 2 |
| 3 | 3543 | 1 |

P.K

F.K

4 ✗

is not allowed in child table bcz not in P.T

Calling PersonID in orders table.

Reference

⇒ Records cannot be inserted into a child table if corresponding record in master table does not exist.

⇒ Record of master table cannot be deleted if corresponding record in detail table actually exists.

⇒ Parent must be unique on primary key.

⇒ child may have duplicate/Null unless it is specified.

⇒ Foreign key maintains referential Integrity.

Referenced table

1) Insert :- No violation
↓
Problem

2) Delete : May cause problem.

use → on delete cascade
on delete set null
on delete NO Action

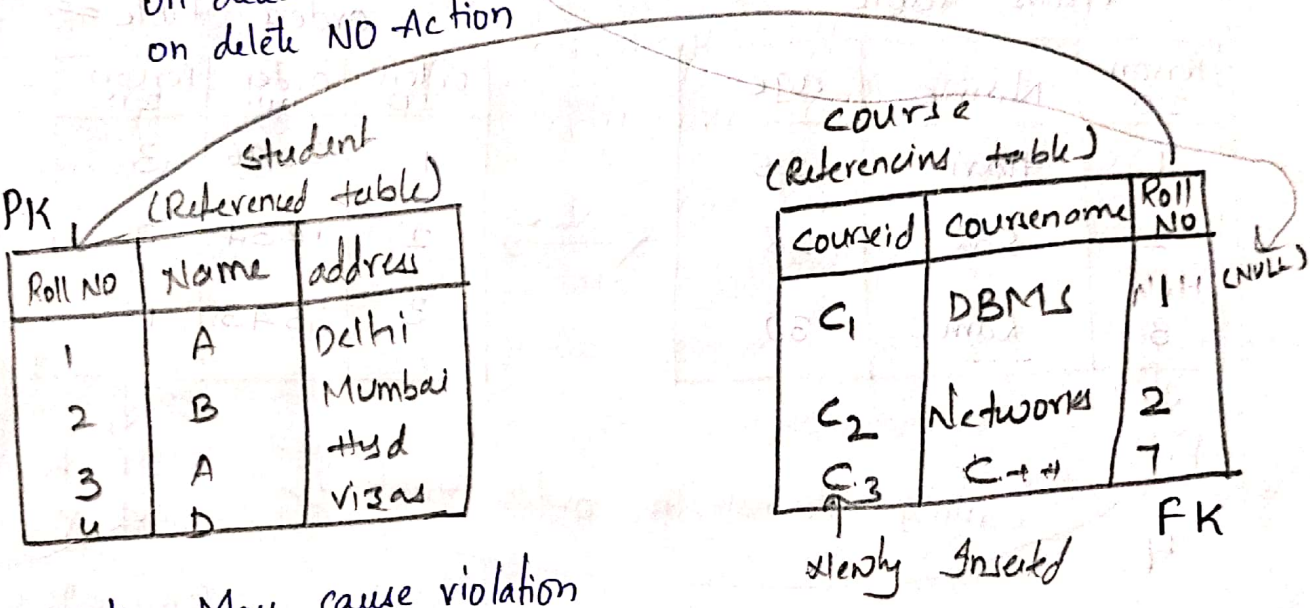
3) Updation: May cause violation

Referencing table

1) Insert : May cause violation

2) Delete : will not cause any violation

3) Updation: May cause violation




3) Updation: May cause violation

~~Compound key~~ = Difference b/w Primary key & foreign key.

| Primary key | Foreign key |
|---|---|
| • Helps you to uniquely identify a record in the table | • It is a field in the table that is the primary key of another table. |
| • Primary key never accept null values | • A ~~primary~~ foreign key may accept multiple null values. |
| • Primary key is a clustered index | • A foreign key cannot automatically create an index. |
| • You can have single primary key in a table | • You can have multiple foreign keys in table. |

Composite key:-

A key which has multiple attributes to uniquely identify rows in table is called composite key.

Difference between Primary key & unique key:

| Primary key | Unique key |
|---|---|
| • Primary key can't accept null values.<br>• We can have only one primary key in a table | • Primary key can accept only one null value.<br>• We can have more than one unique key in table. |

## Foreign key maintains referential integrity:-

Primary key

| Roll NO | Name | address |
|---|---|---|
| 1 | A | Delhi |
| 2 | B | Mumbai |
| 3 | A | Hyd |

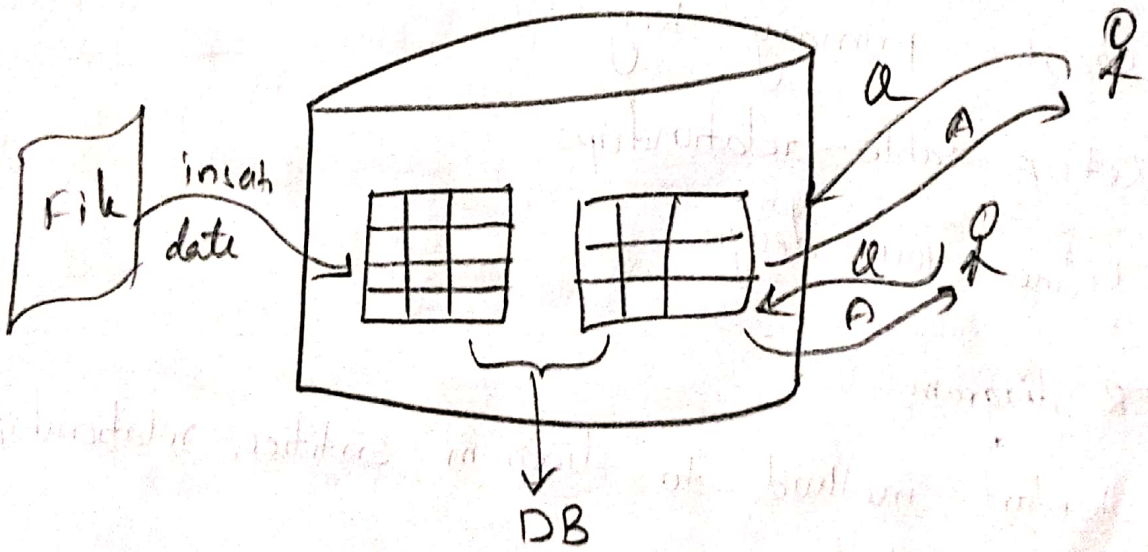
| courseid | coursename | Roll no |
|---|---|---|
| $C_1$ | DBMS | 1 |
| $C_2$ | Java | 2 |
| | | |

Foreign key

## Querying Relational data:

Steps in creating and using a (relational) database

1. Design schema, create using DDL
2. Load initial data
3. Execute querries and modifications.



information content in db

## Logical Database design:-

converting ER diagrams to relational tables.

Logical database design is the process of transforming (or mapping) a conceptual schema of the application domain into a data model underlying a particular DBMS.

Logical databases are special programs that retrieve data and make it available to application programs.

* steps in design process:

1. Determine the purpose of database
2. Find and organize the information required.
3. Divide the information into tables.
4. Turn information items into columns.
5. Specify primary key
6. Setup table relationships
7. Refine your design

ER diagram

→ standard method to diagram Entities, relationships and attributes

→ It is a modelling technique.

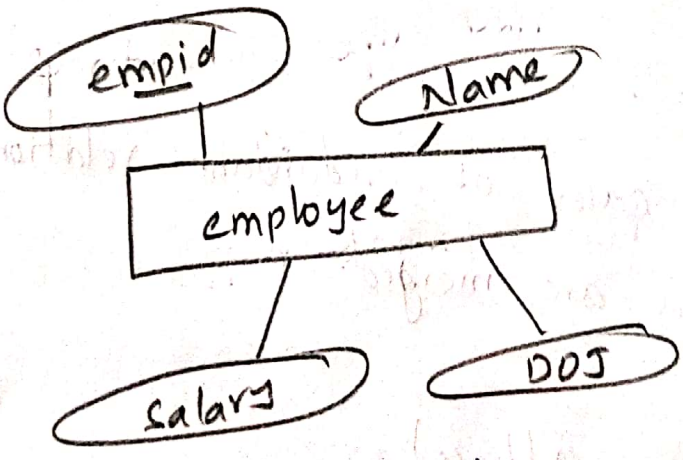
→ If the target database is relational, then it will be mapped on normalized relations.

→ Four steps to develop logical database design.

1. Represent entities
2. Represent relationship
3. Merge relations
4. Normalize relations

Represent Entities. (entity sets to tables)

each Entity of E-R model is represented as relation.




Employee

P.K

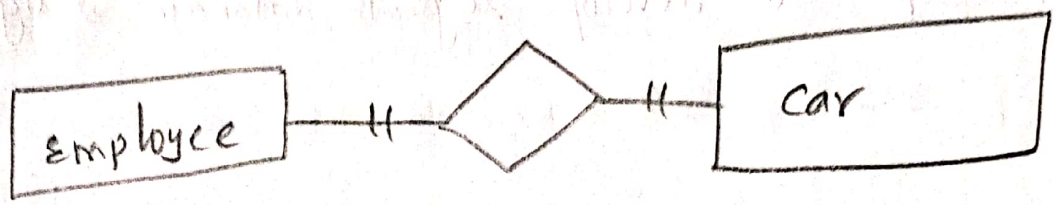
| empid | Name | DOJ | salary |
|---|---|---|---|

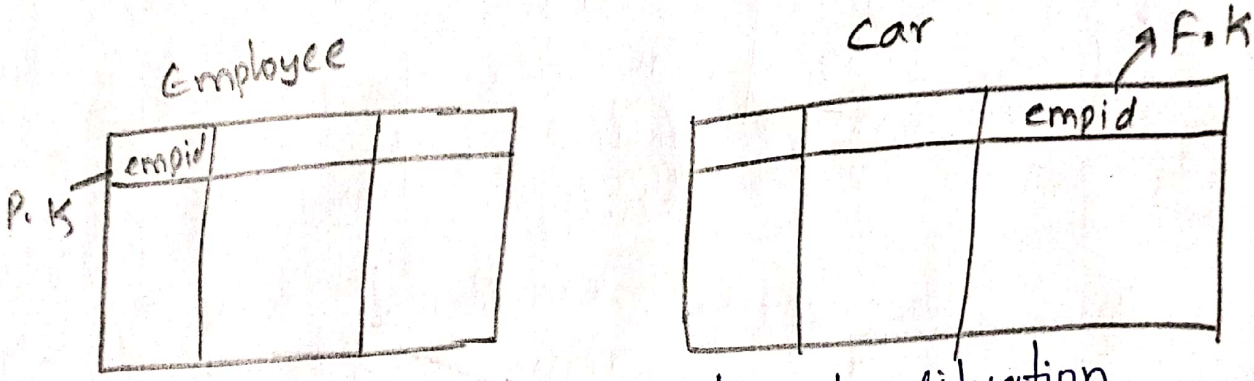
- Name of the Entity becomes name of relation
- Attributes of each Entity becomes the column of relations.
- The field which is used as identifier of the relation is marked as primary key and other attributes remain same.

Represent relationships.

- Each relationship of E-R is also represented in relational model.

eg:-




- Relationship may vary according to situation
- Sometimes relationship can be made using primary and foreign keys.

Merge relations:

- There may be some cases of redundant relations.
- In such cases relations are merged.

eg:-

student (Roll_No, Name, Address)

sudent1 (Roll_No, class, DOB) } merged ↓

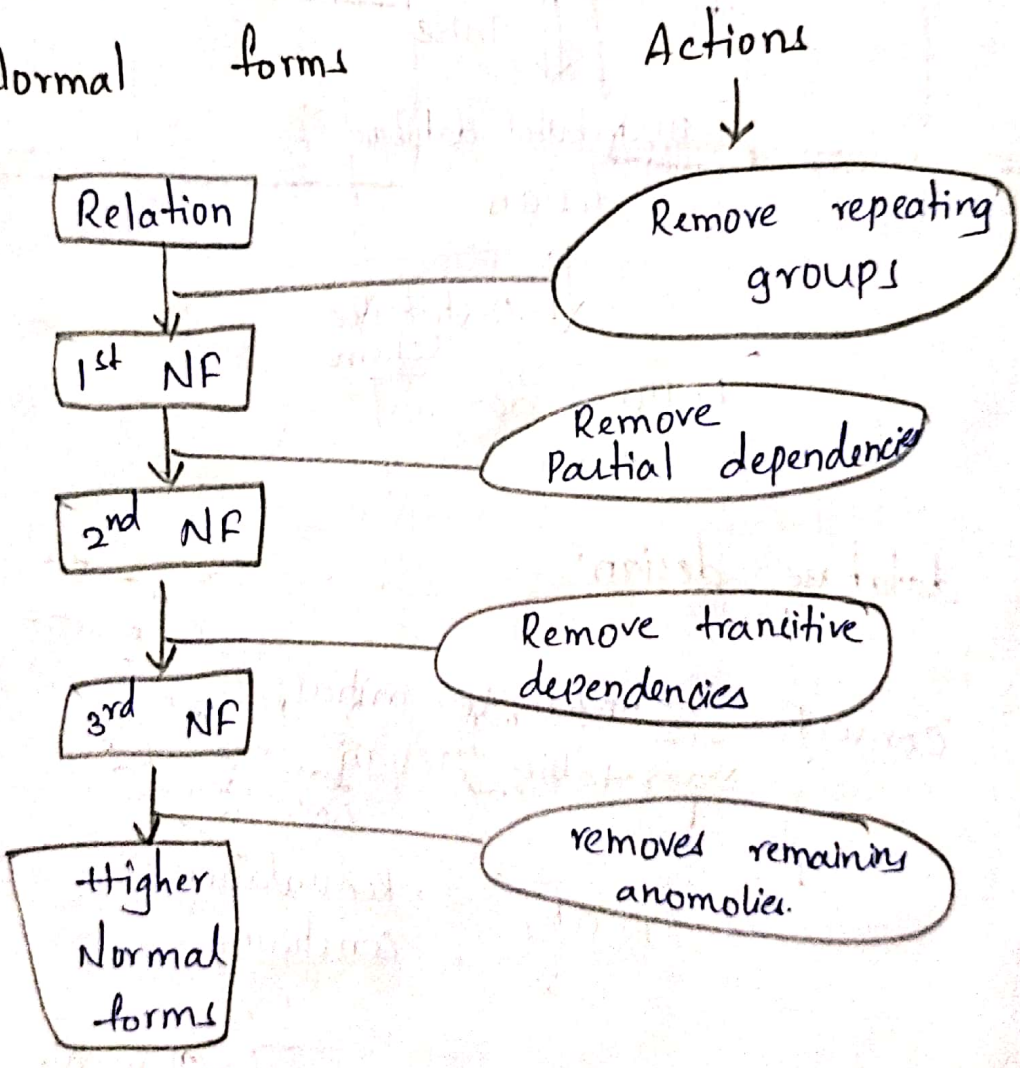
→ student (Roll_No, Name, class, DOB, address)

Normalize Relations:

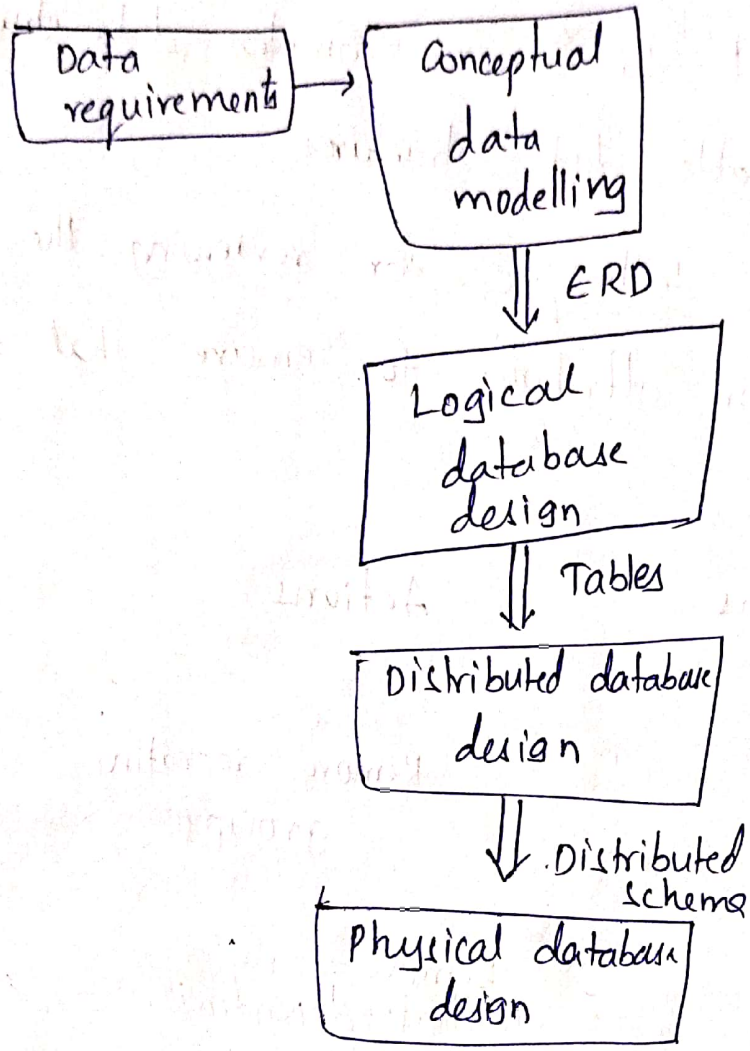
It is the process of efficiently organizing the data in database.

Normalization is used to convert complex data structures into simple and stable data structures.

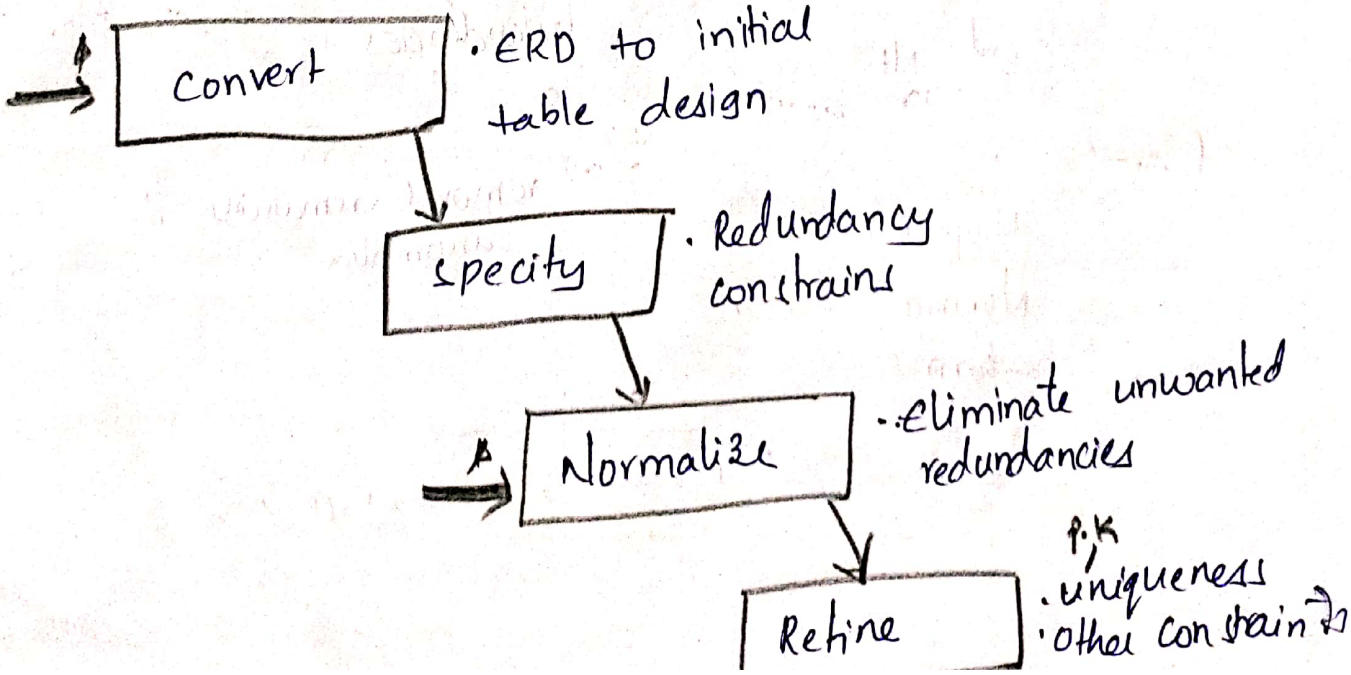
Normalization is a technique for reviewing the list of entities and their attributes to ensure that attributes are stored.



Database development phases.




logical database design:



# Logical database design: ER to relational.

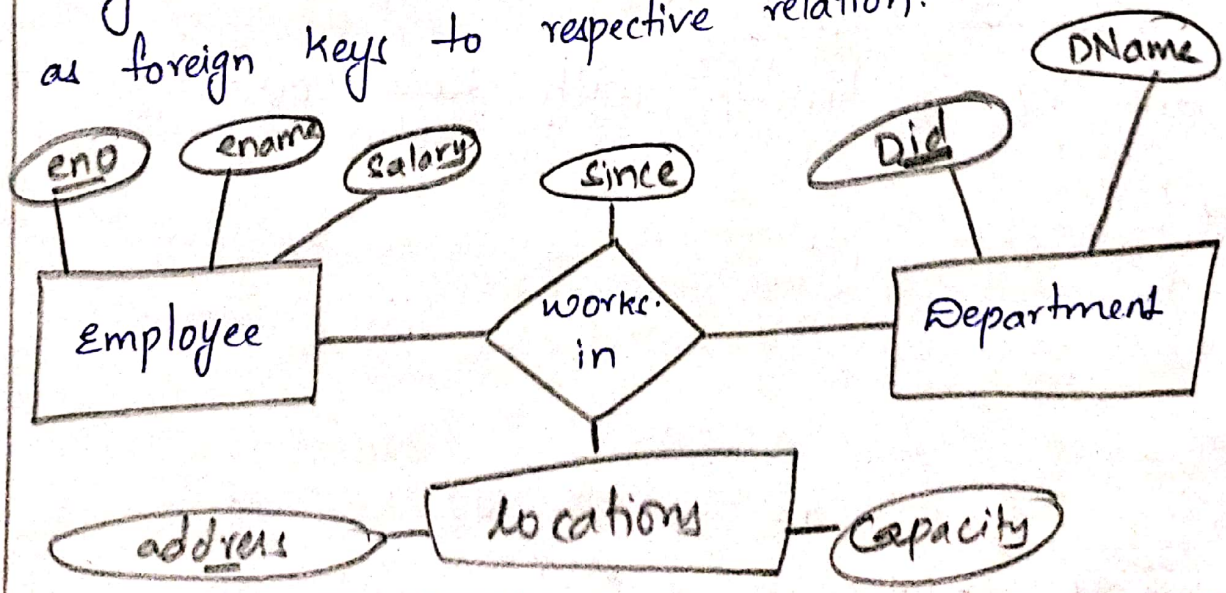
1. Entity sets to Tables
2. Relationship sets to tables
3. Translating relationship sets with key Constraints
4. Translating relationship sets with participation
5. Translating weak entity sets.
6. Translating class Hierarchies
7. Translation of ER diagrams with aggregation.

1. Entity sets to tables:- (already wrote).
SQL: create table (employees ssn char(11), name char(30), Primary key(ssn));

2. Relationship sets to tables:-

A relationship set is mapped into relation, the attributes of the relation includes

- key attributes of participating relation are declared as foreign keys to respective relation.



Employee

P.K

| eno | ename | salary |
|---|---|---|
| 1 | A | 15K |
| 2 | B | 30K |

Department

P.K

| Did | Dname |
|---|---|
| 101 | CSE |
| 102 | IT |

works in

| eno | Did | since |
|---|---|---|
| 1 | 101 | 1-Jan |
| 2 | 102 | 1-feb |

~~Translating~~ ~~relationship~~ ~~sets~~

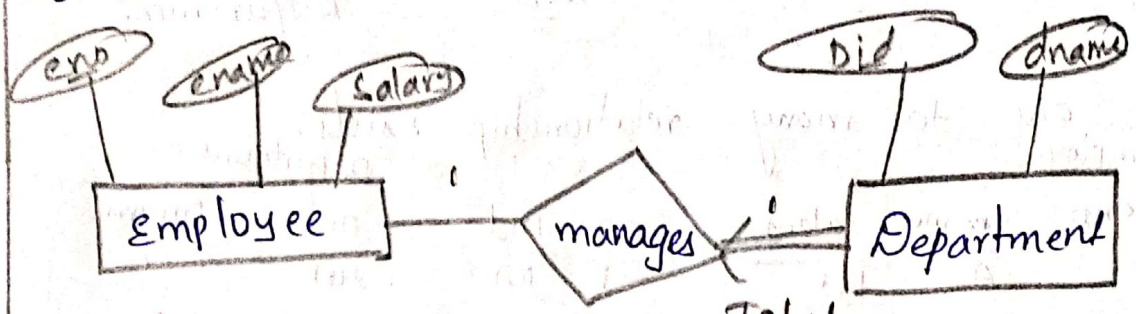
CREATE TABLE works_in (eno char(11), did Integer,
address char(20), since DATE, PRIMARY KEY (eno, did,
address), FOREIGN KEY (eno) REFERENCES Employees,
FOREIGN KEY (address) REFERENCES Locations,
Foreign key (did) REFERENCES Department)

~~Translating~~ ~~relationship~~ ~~with~~ ~~key~~ ~~Constraints~~

Eg:-(2)

## Translating relationship with participation constraints:

- If entity set gets totally participated with relationship set then foreign key with NOT NULL constraints.



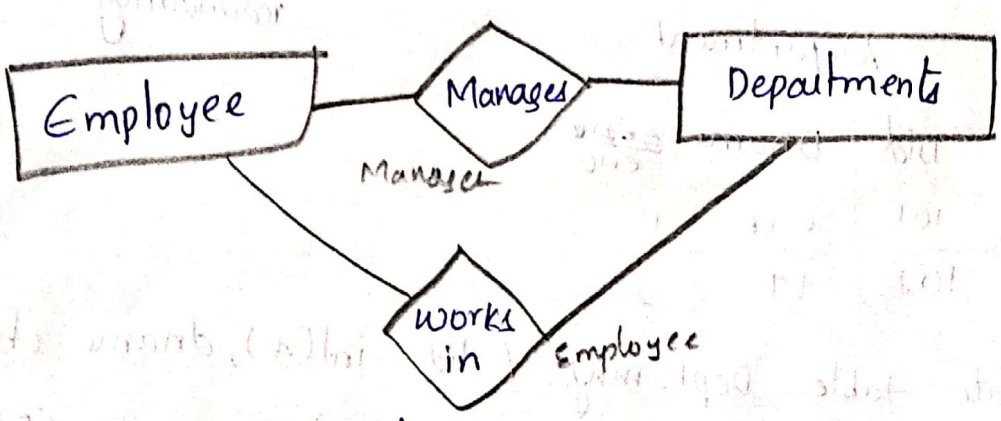

Not all emp. manages dept.

Partial Participation: Not every employee manages.

Total Participation.

↑ Dept. should be managed by someone

→ Instance comes entity comes.



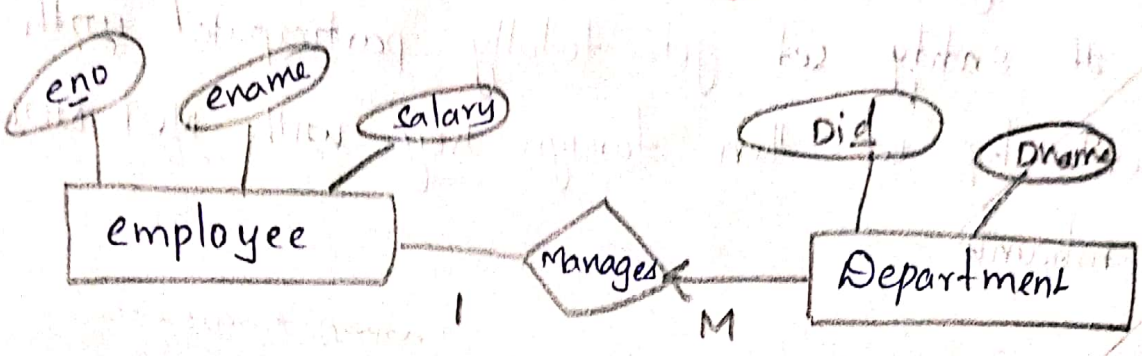

worksin — empid, did, since

Create table dept_mgr (did int(4), dname char(20), eno int(4) NOT NULL, since DATE,

Primary key(did) Foreign key(eno) references Employees ON DELETE NO ACTION)

ON • CASCADE

# Translating relationship with Key Constraints:




one to many relationship exists.

Employee:

| eno | ename | salary |
|---|---|---|
| 1 | A | 15k |
| 2 | B | 19k |
| | | |

Manages

| eno | Did |
|---|---|
| 1 | 101 |
| 2 | 103 |
| | |

Department

| Did | Dname |
|---|---|
| 101 | CSE |
| 102 | IT |
| 103 | ECE |

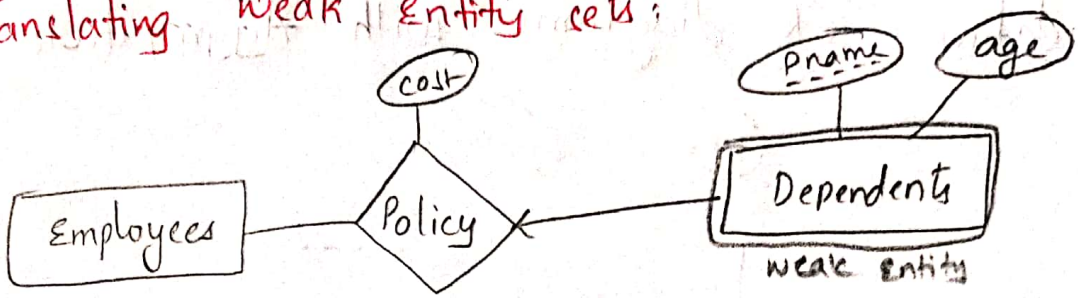
Merge these tables to avoid redundancy.

Department

| Did | Dname | ~~eno~~ eno |
|---|---|---|
| 101 | CSE | 1 |
| 102 | IT | 2 |

Create table Dept_mnger (did int(4), dname char(4),
eno int(3), Primary key(Did),
Foreign key(eno) References
(employee));

## Translating weak entity sets:

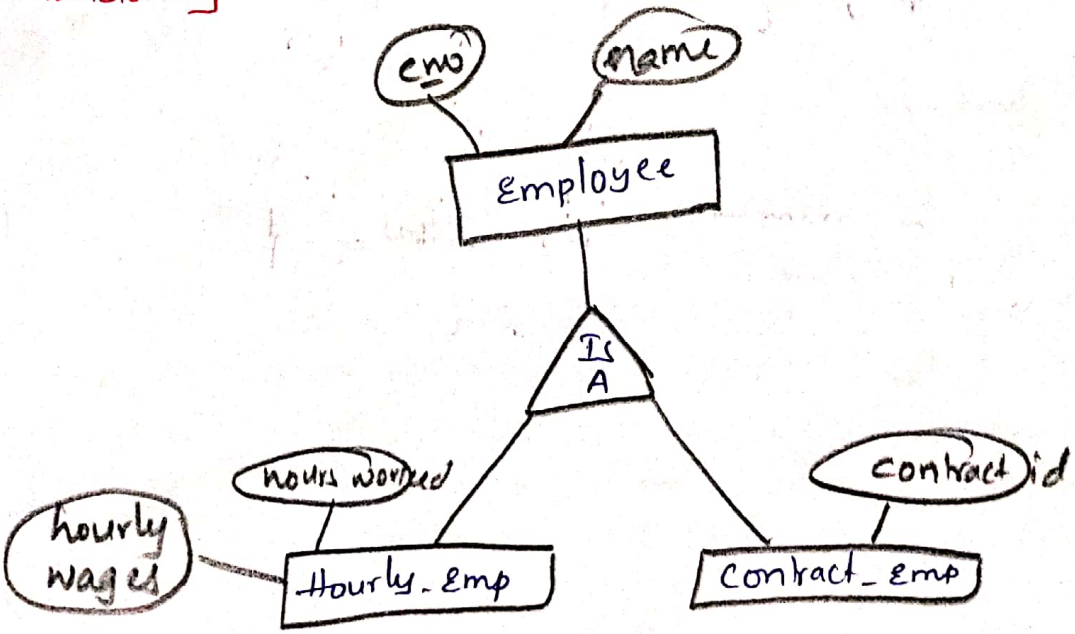



Create table Dep_policy (Pname char(20), age Integer, eno in(4), Primary key (Pname, eno), Foreign key (ssn) References employees ON Delete cascade)

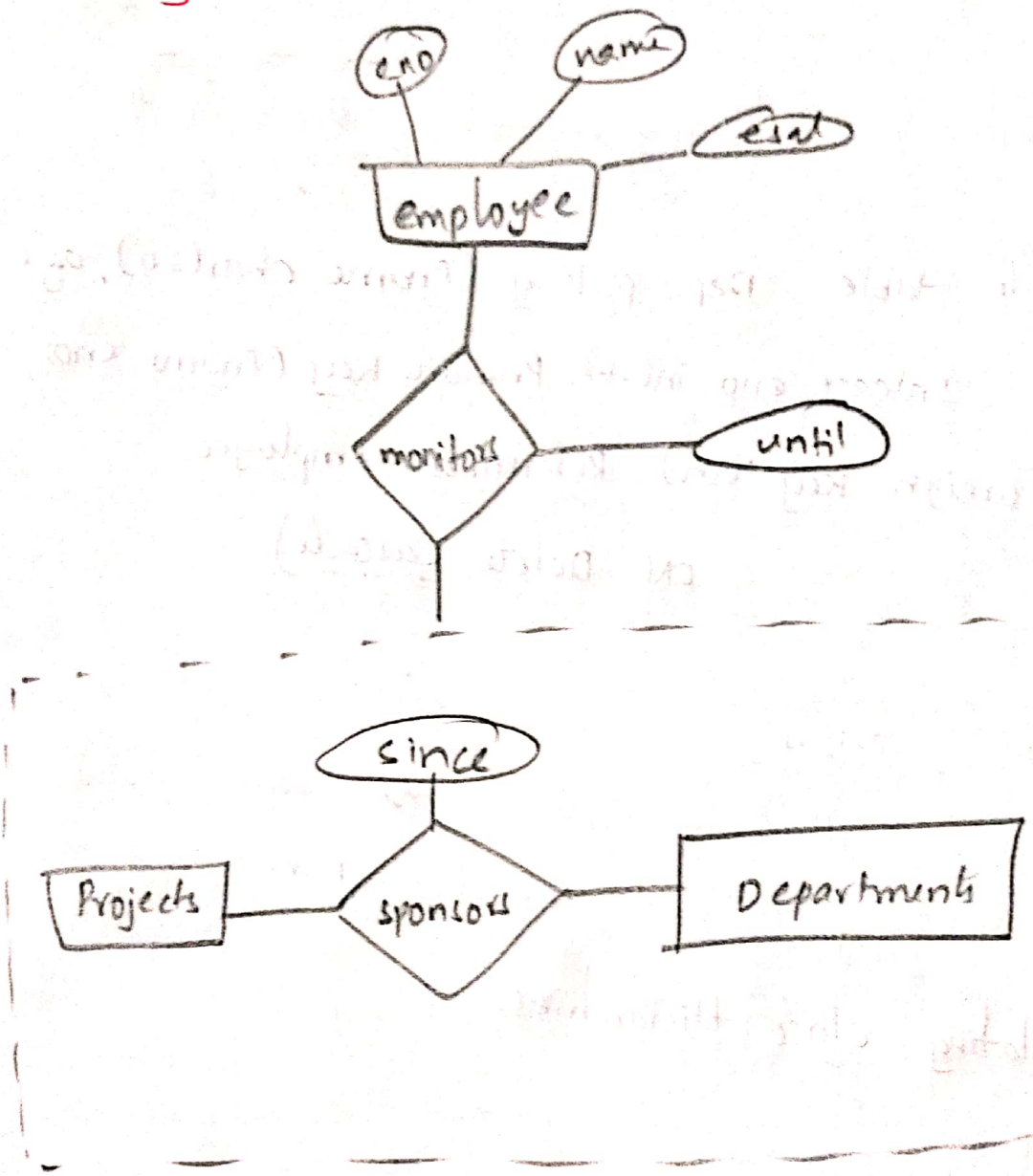
| Emp | Policy |
|---|---|
| 101 | X |

from emp the entire will be deleted both in paunt & child table.

## Translating class Hierarchies:-



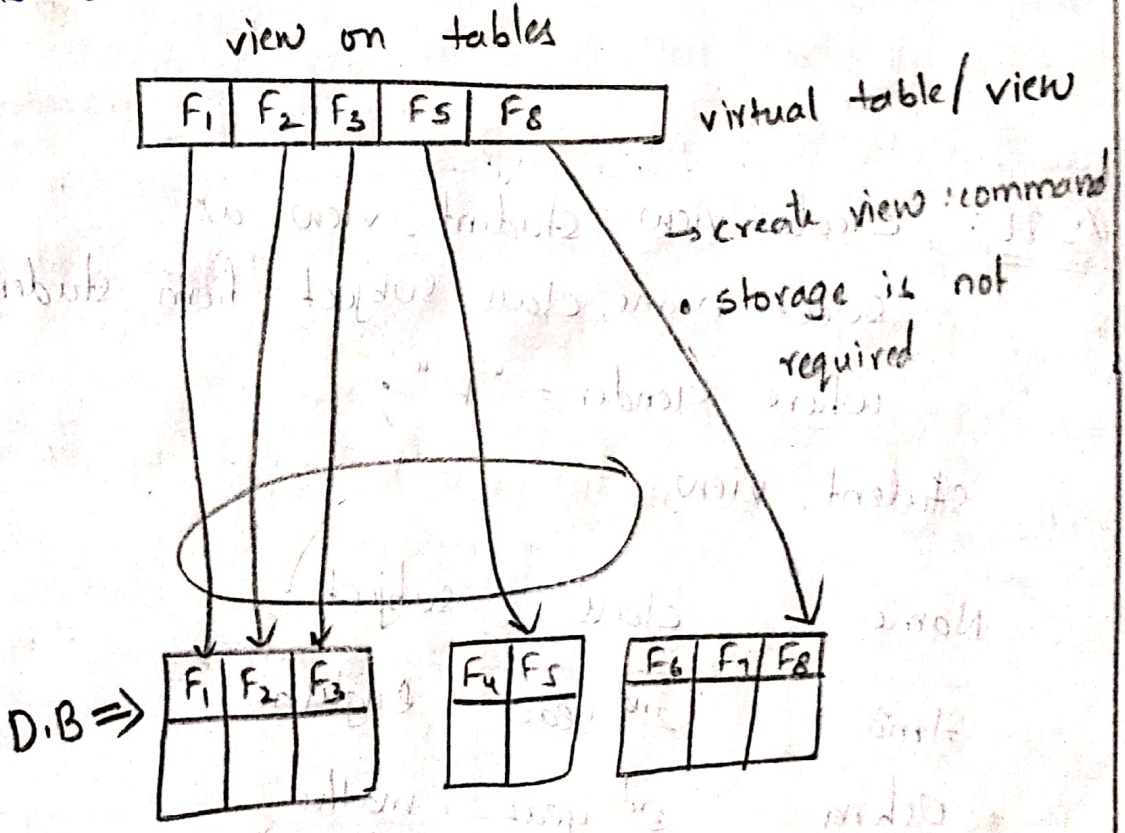
# Translating ER diagrams with Aggregation



## * Introduction to views :

View is a virtual table. It is a logical or virtual table composed of the result set of query. views are created using SQL statements.

- It is a dynamic, virtual table computed from data in database. → when data is updated in table it will be updated in view also
- changing the data in tables alter the data in databases.




Advantages:-

- views can subset the data contained in table
- They can join & simplify tables into a single virtual table.

- views can hide complexity of db.
- views cannot require any extra storage.
- It provides security.

Eg:- student

I/P:

| Roll NO | Name | Gender | class | subject | city |
|---|---|---|---|---|---|
| 1 | Fakhir | M | 1st year | Maths | Hyd |
| 2 | Amna | F | 2nd year | english | Mumbai |
| 3 | Yasir | M | 1st year | computer | Delhi |
| 4 | Ushna | F | 3rd year | maths | LOHAT |
| 5 | Ahad | M | First year | english | Rajastha |

*SQL: create view student_view as
select name, class, subject from student
where Gender = "F";

O/P:- student_view

| Name | class | subject |
|---|---|---|
| Amna | 2nd year | English |
| Ushna | 3rd year | maths. |

Types of views:-
1. Read-only view
2. Updateable view

* Read-only views:- Allows only Select operations
* Updateable views:- Allows Select, insert, update and delete operations

## Updateable views:-

views can also be used for data manipulation. views on which data manipulation can be done are called updateable views.

For a view to be updateable, it should meet the following criteria

1. views defined from single table.
2. If user wants to insert records with the help of view then primary key and not null should be included in view.

eg:- update stu set name = 'xy3' where enroll = 4866;

1 row updated

SQL > Select * from stu;

| | |
|---|---|
| 4866 | XYZ |
| 4546 | BDJG |

## Destroying a view:

Drop command drops the specified view.

syntax:- Drop view viewname

Eg:- Drop view stu;

o/p: view dropped.

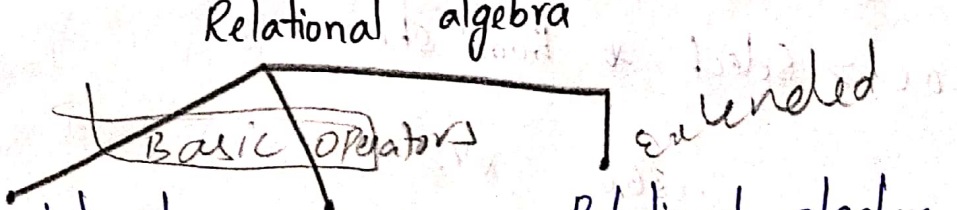
SQL query → Relational algebraic exp → Query execution → executable code

## * Relational Algebra:-

E.F. codd (1970)

like c alg. Relational algebra is a widely used procedural query language. It collects instances of relations as input and gives occurences of relations as output. It uses various operation to perform this action.

Relational algebra operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations.




Unary relational operations
- They operate on 1 relation
eg:- select

Binary relational operators
- Pair of relations
eg:- cartesian product
∩, ∪.

Relational algebra operations from set theory.

Unary Relational Operations:

- SELECT (symbol: $\sigma$) =, ≠, <, ≤, >, ≥
- PROJECT (symbol: $\Pi$) basic
- RENAME (symbol: $\rho$)

Binary Relational operations:

- JOIN ($\bowtie$) — Extended
  - Theta join ($\theta$)
  - Equi join( )
  - Natural join( )
  - Outer join( )
  - Left outer join ($A \bowtie B$)
  - Right " " ($A \bowtie B$)
  - Full outer join ($A \bowtie B$)
- DIVISION ( /, ÷ )

Relational algebra operations from set theory:- (basic Binary)

- UNION (U)
- INTERSECTION ($\cap$)
- DIFFERENCE (-)
- CARTESIAN PRODUCT (X)

## SELECT ($\sigma$):

The SELECT operation is used for selecting a subset of tuples according to a given selection condition.

Sigma ($\sigma$) symbol denotes it.

It is used as an expression to choose tuples which meet selection condition. Select operations selects tuples that satisfy a given predicate.

$\sigma_p(r)$

- $\sigma$ is the predicate
- r stands for relation (table name)
- p prepositional logic → select condition

↳ filtering criteria

## Destroying a view:

Drop command drops the specified view.

Drop view viewname

Eg:- Drop view stu;

view dropped

student

| sid | student |
|---|---|
| 1 | A |
| 2 | B |

$\sigma_{sid=1}$ (student)

o/p:-

| sid | sname |
|---|---|
| 1 | A |

Examples:-

① $\sigma_{topic = "Database"}$ (Tutorials)

o/p:- selects tuples from Tutorials where topic = 'Database'

② $\sigma_{topic = "Database" \text{ and } author = "guru"}$ (Tutorials)

o/p:- selects tuples from tutorials where the topic is 'Database' and author is guru.

③ $\sigma_{sales > 5000}$ (customers)

o/p:- selects tuples from customers where sales is greater than 5000.

Project – columns – ($\pi$)

$\pi_{sname, rating}(S_2)$ (retrieves columns sname, rating from $S_2$)

| sname | rating |
|---|---|
| Yuppy | 9 |
| Lubber | 8 |
| Guppy | 5 |
| Rusty | 10 |

By combining select ($\sigma$) and Projection ($\pi$) ...

$\pi_{sname, rating}(\sigma_{rating > 8}(S_2))$

| sname | rating |
|---|---|
| Yuppy | 9 |
| Rusty | 10 |

Union ($\cup$) :- $S_1 \cup S_2$

| sid | sname | rating | age |
|---|---|---|---|
| 22 | Dustin | 7 | 45.0 |
| 31 | Lubber | 8 | 55.0 |
| 58 | Rusty | 10 | 35.0 |
| 28 | Yuppy | 9 | 35.0 |
| 44 | Guppy | 5 | 35.0 |

Queries:

Sailors (sid: Integer, sname: string, rating: integer, age: real)
(Key)

Boats (bid: Integer, bname: string, colour: string)
(Key)

Reserves (sid: Integer, bid: integer, day: date)

all are keys

Sailors: $S_1$

| sid | Sname | rating | age |
|---|---|---|---|
| 22 | Dustin | 7 | 45.0 |
| 31 | Lubber | 8 | 55.0 |
| 58 | Rusty | 10 | 35.0 |

Sailors: $S_2$

| sid | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35 |
| 31 | lubber | 8 | 55 |
| 44 | guppy | 5 | 35 |
| 58 | rusty | 10 | 35 |

Reserves: R1

| sid | bid | day |
|---|---|---|
| 22 | 101 | 10/10/96 |
| 58 | 103 | 11/12/96 |

Select – rows – ($\sigma$)

$\sigma_{rating>8}(S_2)$ (retrieves rows from $S_2$ whose rating > 8.)

| sid | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35.0 |
| 58 | Rusty | 10 | 35.0 |

Division (/):

O/P:-

A

| sno | pno |
|---|---|
| $S_1$ | $P_1$ |
| $S_1$ | $P_2$ |
| $S_1$ | $P_3$ |
| $S_1$ | $P_4$ |
| $S_2$ | $P_1$ |
| $S_2$ | $P_2$ |
| $S_3$ | $P_2$ |
| $S_4$ | $P_2$ |
| $S_4$ | $P_4$ |

B1

| pno |
|---|
| $P_2$ |

A/B1

| sno |
|---|
| $S_1$ |
| $S_2$ |
| $S_3$ |
| $S_4$ |

B2

| pno |
|---|
| $P_2$ |
| $P_4$ |

A/B2

| sno |
|---|
| $S_1$ |
| $S_4$ |

B3

| pno |
|---|
| $P_1$ |
| $P_2$ |
| $P_4$ |

A/B3

| sno |
|---|
| $S_1$ |

Intersection ($\cap$): $S1 \cap S2$

| sid | sname | rating | age |
|---|---|---|---|
| 31 | Lubber | 8 | 55.5 |
| 58 | Rusty | 10 | 35.0 |

Difference (−): $S1 - S2$

| sid | sname | rating | age |
|---|---|---|---|
| 22 | Dustin | 7 | 45.0 |

Cartesian (×): $S1 \times R1$

| sid | sname | rating | age | (sid) | bid | day |
|---|---|---|---|---|---|---|
| 22 | Dustin | 7 | 45.0 | 22 | 101 | 10/10/96 |
| 22 | Dustin | 7 | 45.0 | 58 | 103 | 11/12/96 |
| 31 | Lubber | 8 | 55.0 | 22 | 101 | 10/10/96 |
| 31 | Lubber | 8 | 55.0 | 58 | 103 | 11/12/96 |
| 58 | Rusty | 10 | 35.0 | 22 | 101 | 10/10/96 |
| 58 | Rusty | 10 | 35.0 | 58 | 103 | 11/12/96 |

Equi join: (=): $S1 \underset{R.sid = sid}{} R1$

| sid | sname | rating | age | bid | day |
|---|---|---|---|---|---|
| 22 | Dustin | 7 | 45.0 | 101 | 10/10/96 |
| 58 | Rusty | 10 | 35.0 | 103 | 11/12/96 |

Examples:-

student

| sid | sname | Teacher | marks |
|---|---|---|---|
| 1 | A | $T_1$ | 90 |
| 2 | B | B | 80 |
| 3 | C | $T_1$ | 85 |
| 4 | D | B | 75 |

① Query:- Details of student whose name is same as teacher.

$\sigma_{Sname = Teacher}$ (student)

SQL >

o/p:-

| 2 | B | B | 80 |
|---|---|---|---|

② Details of student whose marks are greater than or equal to 85.

$\sigma_{(marks \geq 85)}$ (student)

o/p:-

| 1 | A | $T_1$ | 90 |
|---|---|---|---|
| 3 | C | $T_1$ | 85 |

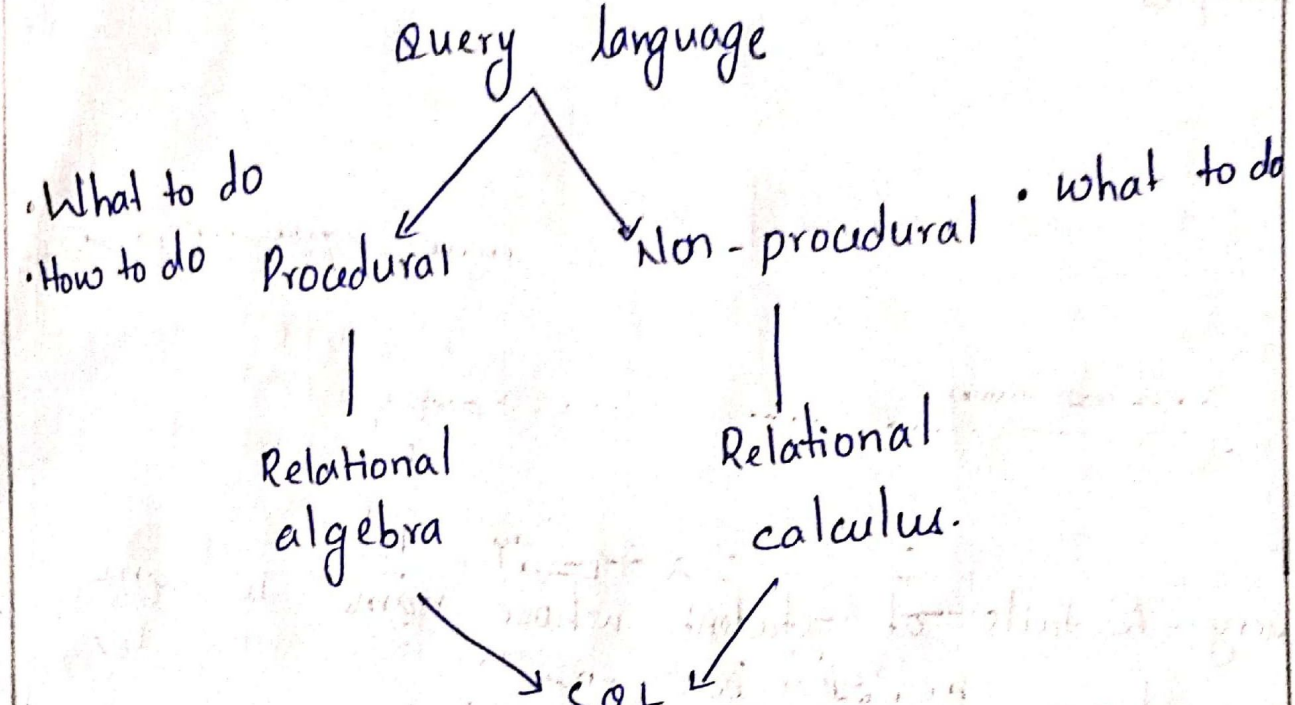



Table : customer

| customer_id | customer name | customer city |
|---|---|---|
| C101 | Sai | agra |
| C102 | Raghu | Agra |
| C103 | chaitanya | Noida |

Query :-

$$\sigma_{cust\_city = \text{"agra"}}(customer)$$

o/p :-

| customer_id | customer name | customer city |
|---|---|---|
| C101 | Sai | agra |
| C102 | Raghu | Agra |

## Projection ($\pi$):

It yields columns (attributes) of a relation. Using project operation duplicate values are automatically removed.

syntax:- $\pi_{A1, A2, \ldots An}(R)$

- $A1, A2, \ldots An$ → name of attributes (column names)
- $R$ → Relation

Example:- student

| Roll No | Sname | class |
|---|---|---|
| 1 | A | 11 |
| 2 | B | 12 |
| 3 | C | 11 |
| 4 | B | 12 |

Display name of students who studies in class 12

$\pi_{name}(student)$

o/p:- A
B → one value is removed (2 times appeared)
C

$\pi_{name}[\sigma_{class=12}(student)]$

↓ complete relation where class = 12

o/p:- B

Table: customer

| customer_id | customer_name | customer_city |
|---|---|---|
| c101 | sai | agra |
| c102 | raghu | agra |
| c103 | chaitanya | Noida |
| c104 | Ajeet | Delhi |
| c105 | carl | Delhi |

Query:-

$\Pi_{cust\_name,\ customer\_city}$ (CUSTOMER)

o/p:-

| Customer name | customer city |
|---|---|
| sai | agra |
| raghu | agra |
| chaitanya | Noida |
| Ajeet | Delhi |
| carl | Delhi |

Project operator in relational algebra is similar to select statement in SQL.

## Rename (ρ)

It is a unary operation used for renaming attributes of a relation, or a relation.

symbol – (rho) → ρ

syntax:- ρ (new relation name, old relation name).

| customer_ID | customer name | customer city. |
|---|---|---|
| c101 | Sai | Agra |
| c102 | Ram | Noida |
| c103 | Sita | Agra |
| c104 | Shiva | Delhi |
| c105 | Ganga | chennai |

query:- ρ (cust_names, π(customer_name) (customes))

- cust_names ↓ New Column name
- customer_name ↓ old column name
- customes ↓ Table name

output:-

| cust_names |
|---|
| Sai |
| Ram |
| Sita |
| Shiva |
| Ganga |

## Set Operations:-

1. **Union (∪)**: (symbol) It is used to select all the rows (tuples) from two tables (relations).

Lets say we have two relations R and S both have same columns and we want to select all tuples (rows) from these relations then we can apply union operator on this i.e R ∪ S

- Two relations are said to be union if the following conditions hold.

→ They have same number of fields

→ corresponding fields taken inorder from left to right have the same domains.

**Syntax:-** Tablename 1 ∪ Tablename 2

Table A

| column 1 | column 2 |
|---|---|
| 1 | 1 |
| 1 | 2 |

Table B

| column 1 | column 2 |
|---|---|
| 1 | 1 |
| 1 | 3 |

This has been removed due to duplication.

Table A ∪ B

| column 1 | column 2 |
|---|---|
| 1 | 1 |
| 1 | 2 |
| 1 | 3 |

Table 1 : course

| course ID | student name | student ID |
|---|---|---|
| C101 | sai | S501 |
| C104 | Teja | S501 |
| C105 | Ram | S911 |
| C109 | Sita | S921 |

Table 2 : student

| student Id | student name | student age |
|---|---|---|
| S501 | sai | 19 |
| S501 | Teja | 18 |
| S911 | Ram | 17 |
| S921 | Sita | 18 |

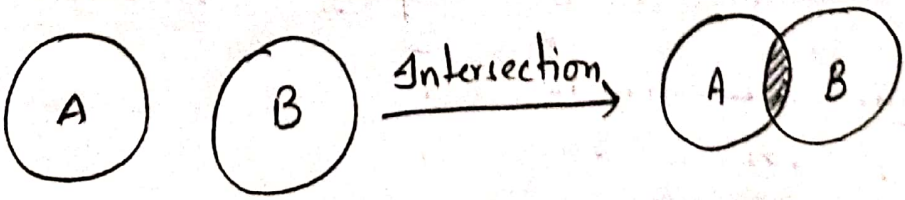
$\Pi$ student_name (course) $\cup$ $\Pi$ student_name (student)

output :-

| student_name |
|---|
| sai |
| Teja |
| ram |
| Sita. |

Intersection ($\cap$) :-

An intersection is defined by the symbol '$\cap$'. $A \cap B$. Defines a relation consisting of a set of all tuples that are in both A and B. However, A and B must be union-Compatible.



It is used to select common rows (tuples) from two tables (relations).

Let's say we have two relations $R_1$ and $R_2$ both have same columns and we want to select all those tuples that are present in both the relations, then in that case we can apply intersection operation on these two relations $R_1 \cap R_2$.

* Only those rows that are present in both the tables will appear in result set.

Table A

| column 1 | column 2 |
|---|---|
| 1 | 1 |
| 1 | 2 |

Table B

| column 1 | column 2 |
|---|---|
| 1 | 1 |
| 1 | 3 |

A ∩ B

| column 1 | column 2 |
|---|---|
| 1 | 1 |

## Set Difference (−):

Let's say we have two relations R1 and R2 and we want to select all those tuples (rows) that are present in relation R1 but not Present in relation R2, this can be done using Set difference R1 − R2.

Syntax:- Tablename 1 − Tablename 2

- The attribute name of A has to match with attribute name in B.
- The two operand relations A and B should be either compatible or union compatible.

Table A

| column 1 | column 2 |
|---|---|
| 1 | 1 |
| 1 | 2 |

Table B

| Column 1 | column 2 |
|---|---|
| 1 | 1 |
| 1 | 3 |

A − B

| column 1 | column 2 |
|---|---|
| 1 | 2 |

## Cartesian Product (X): /cross

Cartesian product is denoted by X. Let say we have two relations $R_1$ and $R_2$ then the cartesian product of these two relations ($R1 \times R2$) would combine each tuple of first relation R1 with each tuple of second relation $R_2$.

Syntax:- R1 X R2

Table 1 : Student

| Name | roll no |
|---|---|
| A | 1 |
| B | 2 |
| C | 3 |

Table 2 : Grade

| Roll no | Grade |
|---|---|
| 1 | A |
| 2 | B |
| 3 | C |

Query : Name of students having Grade A

$\Pi_{name}(\sigma_{Grade=A}(Student \times Grade))$

only column name — filtering condition

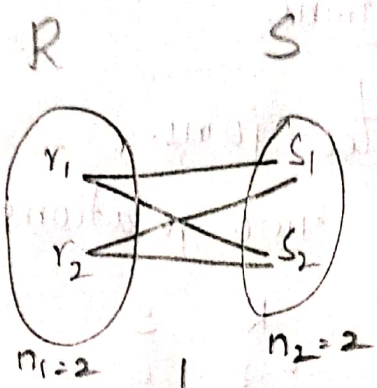
| A | B |
|---|---|
| B | 2 |
| C | 3 |

3 of B * 3 of Y = 9 tuples.

| X | Y |
|---|---|
| Y | 10 |
| Z | 11 |

cross product / cartesian product :-

→ Returns all possible combinations of tuples.




• costly operation

$R \times S$ $\quad 2 \times 2$ $\quad n_1 \times n_2 = 4$

| | |
|---|---|
| $r_1$ | $s_1$ |
| $r_1$ | $s_2$ |
| $r_2$ | $s_1$ |
| $r_2$ | $s_2$ |

$R_1$ $\quad R_2$ $\quad R_1 \times R_2$

Attributes $\quad x \quad y \quad (x+y)$

tuples $\quad n_1 \quad n_2 \quad (n_1 \times n_2)$

R

| A | B |
|---|---|
| $\alpha$ | 2 |
| $\beta$ | 1 |

S

| C | D |
|---|---|
| x | 101 |
| y | 102 |

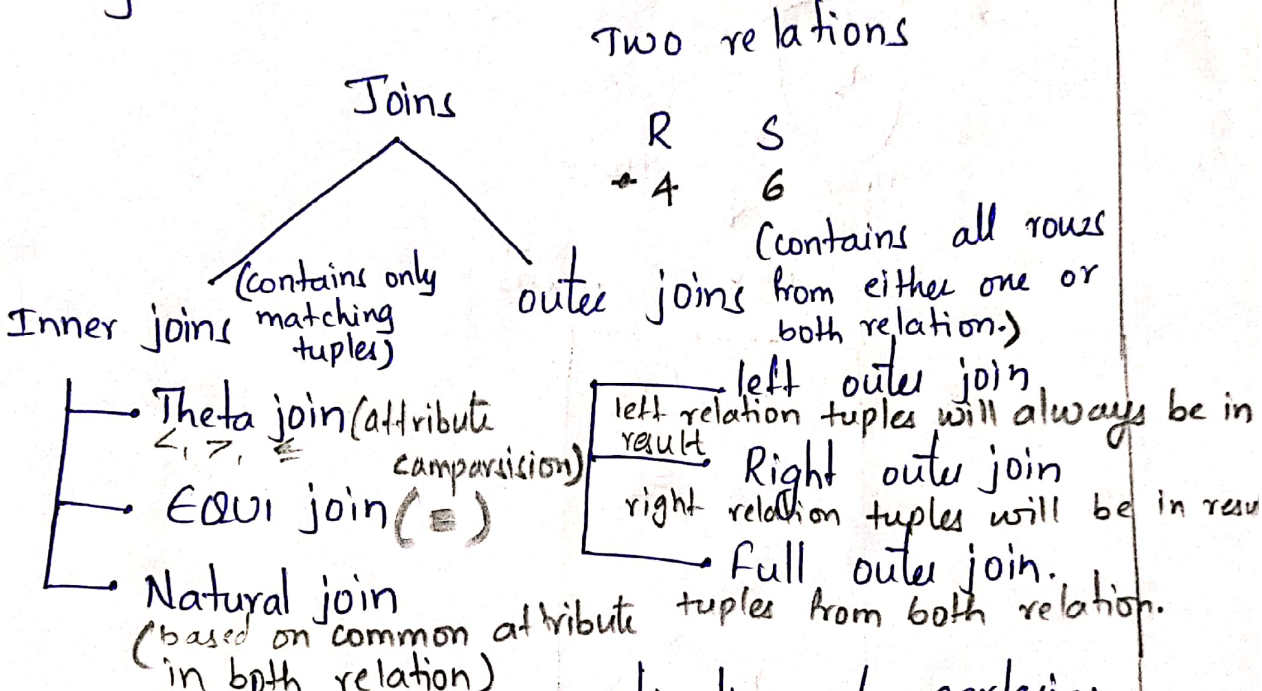
$R \times S$ ⟹

| A | B | C | D |
|---|---|---|---|
| $\alpha$ | 2 | x | 101 |
| $\alpha$ | 2 | y | 102 |
| $\beta$ | 1 | x | 101 |
| $\beta$ | 1 | y | 102 |

# JOINS:

Join operation allows joining variously related tuples from different relations.

Inner joins and outer joins.




Join operation is a combination of cartesian and selection operations.

$R(\overset{3}{A, B, C})$ — ① relation
$S(\overset{2}{E, D})$ — ② relation.
} Join = 3 + 2 = 5

Syntax :- $R \bowtie_{(\text{Join condition})} S$

Join is used to combine related tuples from two relations into single tuple.

→ It is denoted by '⋈'.

## Natural join:- (⋈)

Natural join can be performed if there is a common attribute (column) between the relations.

student relation

| SNO | sname | sage |
|---|---|---|
| $S_1$ | $n_1$ | 21 |
| $S_2$ | $n_2$ | 22 |
| $S_3$ | $n_3$ | 23 |
| $S_4$ | $n_4$ | 24 |

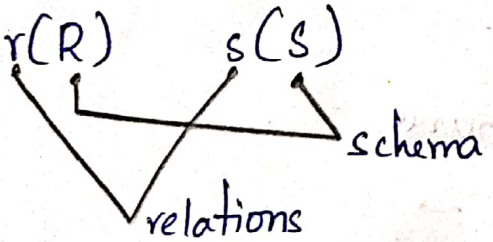
course relation

| sno | cname | fees |
|---|---|---|
| $S_1$ | $C_1$ | 10,000 |
| $S_3$ | $C_3$ | 15,000 |
| $S_5$ | $C_5$ | 20,000 |

→ common attributes in both relation is sno.

→ common " with same values.

Student ⋈ course - natural join




$$r \bowtie S = \Pi_{R \cup S} \left( \sigma_{r.A_2 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \dots s.A_n} (r \times S) \right)$$

↓

common attributes in relation

O/P for natural join:

| sno | sname | sage | cname | fees |
|---|---|---|---|---|
| $s_1$ | $n_1$ | 21 | $c_1$ | 10,000 |
| $s_3$ | $n_3$ | 23 | $c_3$ | 15,000 |

$s_1$ & $s_3$ are common in both.

sno → is common in both.

O/P for left outer join:

Student ⟕ course

| sno | sname | sage | cname | fees |
|---|---|---|---|---|
| $s_1$ | $n_1$ | 21 | $c_1$ | 10,000 |
| $s_2$ | $n_2$ | 22 | null | null |
| $s_3$ | $n_3$ | 23 | $c_3$ | 15,000 |
| $s_4$ | $n_4$ | 24 | null | null |

O/P for right outer join:-

student ⟖ course.

| sno | sname | sage | cname | fees |
|---|---|---|---|---|
| $s_1$ | $n_1$ | 21 | $c_1$ | 10,000 |
| $s_3$ | $n_3$ | 23 | $c_3$ | 15,000 |
| $s_5$ | null | null | $c_5$ | 20,000 |

o/p for full outer join:-

student ⟗ course

| sno | sname | sage | cname | fees |
|---|---|---|---|---|
| $s_1$ | $n_1$ | 21 | c1 | 10,000 |
| $s_2$ | $n_2$ | 22 | null | null |
| $s_3$ | $n_3$ | 23 | $c_3$ | 15,000 |
| $s_4$ | $n_4$ | 24 | null | null |
| $s_5$ | null | null | $c_5$ | 20.000 |

# Relational calculus:-

Relational calculus is a non-procedural query language. It uses mathematical predicate calculus. It provides only the description of query but it does not provide the methods to solve it. What data to be retrieved but not how to do it.

Relational calculus

- Tuple Relational calculus (TRC)
- Domain relational calculus (DRC)

## Tuple Relational calculus:-

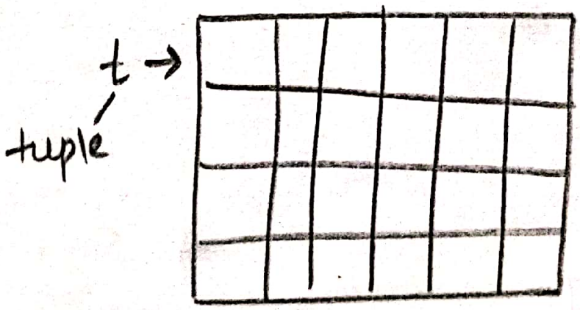
t → Tuple relational calculus (Relational DB) → Predicate calculus.

Tuple relational calculus is used for selecting those tuples that satisfy the given condition.

Syntax:- { T / condition }

{ T / P(T) }

- T → Resulting tuples
- P(T) → condition used to fetch T




tuple variable ⇒ t , Relation ⇒ R

t.A ⇒ column A of tuple t.

- variables in TRC takes tuples (rows)
- " " DRC " fields (attributes)

$\{t / P(t)\}$

It is the set of all tuples t such that Predicate P is true for t.

Syntax for TRC queries:-

1. $R \in Rel$
   ↳ table

   Eg:- $S \in$ sailors.

2. R.a op S.b ⇒ comparision

   Eg:- S.rating = R.rating

3. R.a operator constant.

   Eg:- S.rating > 10

Any atomic formula.

$\neg P$ , $P \wedge q$, $P \vee q$, $P \Rightarrow q$

* $\exists R\ P(R)$
  - $\exists R$ → tuple variable.
  - $P(R)$ → formula
  - $\exists$ → There exists

Eg:- $\exists s\ [s \in$ sailors$]$

* $\forall R\ P(R)$

Example:-

| first_name | last_name | age |
|---|---|---|
| Ajeet | singh | 30 |
| chaitanya | singh | 31 |
| Rajeev | Bhatia | 27 |
| carl | Pratap | 28 |

Query:-

To display last name of students where age is greater than 30.

$\{t.Last\_name \mid student(t) \text{ and } t.age > 30\}$

In this you can see two parts seperated by | symbol. The second part is where we define the condition and in first part we specify the fields which we want to display for selected tuples.

Result:-

Last_name

singh.

Sailors : S3

| sid | sname | rating | age |
|---|---|---|---|
| 22 | Dustin | 7 | 45.0 |
| 29 | Brutus | 1 | 33.0 |
| 31 | Lubber | 8 | 55.5 |
| 32 | Andy | 8 | 25.5 |
| 58 | Rusty | 10 | 35.0 |
| 64 | Horatio | 7 | 35.0 |
| 71 | Zorba | 10 | 16.0 |
| 74 | Horatio | 9 | 35.0 |
| 85 | Art | 3 | 25.5 |
| 95 | Bob | 3 | 63.5 |

Reserves : R2

| sid | bid | day |
|---|---|---|
| 22 | 101 | 10/10/98 |
| 22 | 102 | 10/10/98 |
| 22 | 103 | 10/8/98 |
| 22 | 104 | 10/7/98 |
| 31 | 102 | 11/10/98 |
| 31 | 103 | 11/6/98 |
| 31 | 104 | 11/12/98 |
| 64 | 101 | 9/5/98 |
| 64 | 102 | 9/8/98 |
| 74 | 103 | 9/8/98 |

Boats : B1

| bid | bname | color |
|---|---|---|
| 101 | Interlake | blue |
| 102 | Interlake | red |
| 103 | clipper | green |
| 104 | Marine | red |

Find all sailors with rating above 7.

$\{ S \mid S \in \text{sailors} \wedge S.\text{rating} > 7 \}$

o/p:

| sid | sname | rating | age |
|---|---|---|---|
| 31 | | | |
| 32 | | | |
| 58 | | | |
| 71 | | | |