Database Management System (DBMS)

- DBMS
- Data Warehouse / Data Mart
- Data Mining
- Data Science
  - AI
  - ML
  - PR

Data
- Facts
- Statistics

} for references and analysis

File Management:

Problem:

- Store the student's personal information
- Subject studies
- Marks obtained

```
student struct {
    int Roll-no;
    char[30] name;
    int phone-no;
    char[30] email-id;
    .
    .
    .
    char[20] stream;
}
```

Problems that might be faced:

- Redundancy
- Inconsistency
- Isolation
- Security
- Concurrent Usage
- Atomicity / Transaction Management
- Backup / Recovery
- Constrained

```
st_marks {
    Roll no;
    name
    PA
    email
    sub1
    sub2
    .
    .
    sub4
}
```

Repeated → Roll no, name, PA, email

Subject marks → sub1, sub2, ..., sub4

- **Redundant Data**
  - Wastage of Space
  - Inconsistant Data

- Storing same data in different places may result in inconsistent data.
- Data Isolation: Data is stored in different formats.

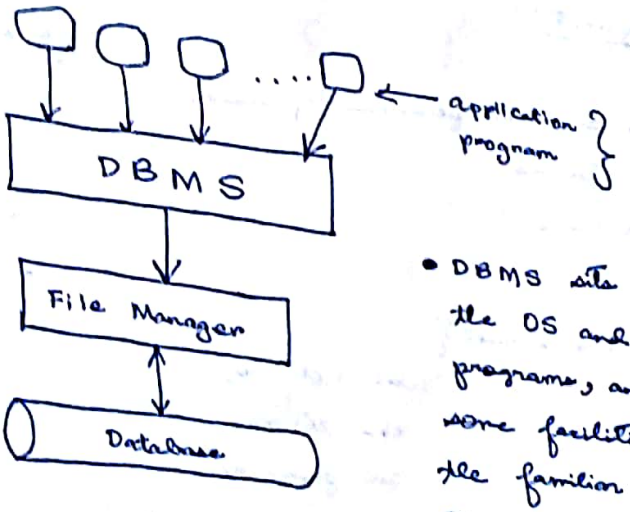
Database Management System:

DBMS consists of—

(DDL) 1) Interrelated data (Database)

(DML) 2) Set of programs that manipulate the data.

- DDL → Data Definition Language
- DML → Data Manipulation Language



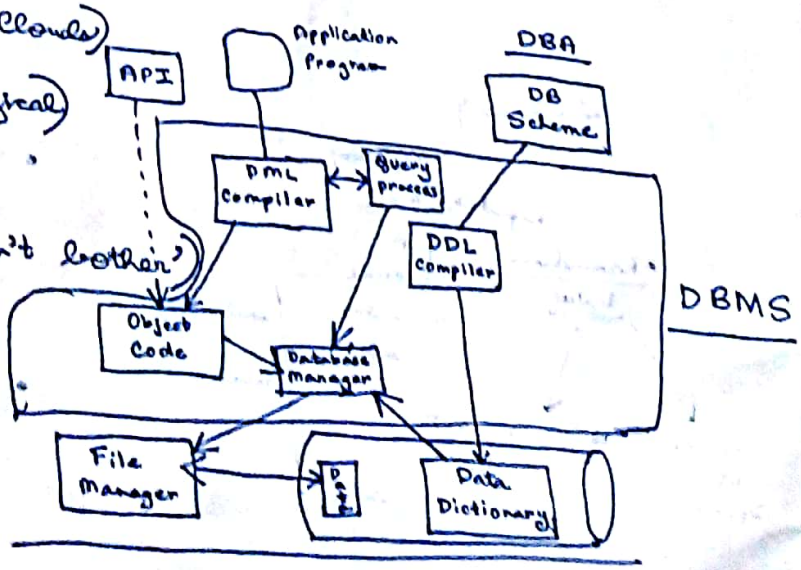
- DBMS sits in between the OS and the application programs, and it provides some facilities to avoid the familiar issues of File Management System.

DBA { 1) Physical Level (clouds)

Programmer { 2) Conceptual (Logical)

User { 3) View

4) End users ('don't bother')



Database Models

1) Entity-Relationship Data Model (E-R Model)
2) Relational Data Model
3) Network Data Model (Not Used Nowadays)
4) Hierarchical Model

(2), (3), (4) → Record Based, ~~Object Based~~

5) Object Oriented Model
6) Object Relational Model
7) Functional Data Model

(5), (6), (7) → Object Oriented

(1) + (Record Based, Object Based) + (Object Oriented) → Object Based

Relation

Function between two variables —
- $m: R \rightarrow R \times R$
- $R: A \rightarrow B$

Function:
- one-to-one *
- one-to-many
- many-to-one *
- many-to-many

DBMS is a software consisting of a collection of interrelated data and a set of programs to access and modify the data.

- database — define || DDL
- Retrieval and processing programs

DML:
- Insert
- Delete
- Update
- Read/Select/Retrieve

2) Relation Model:

users

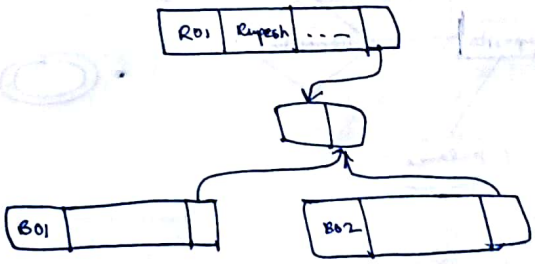
user_id	name	phone	card_id
R01	Rupesh	- - -	- -
R02	Arsh	- - -	- -

Book_id	title	Author
B01	DBMS	Korth
B02	DS	Swapnil

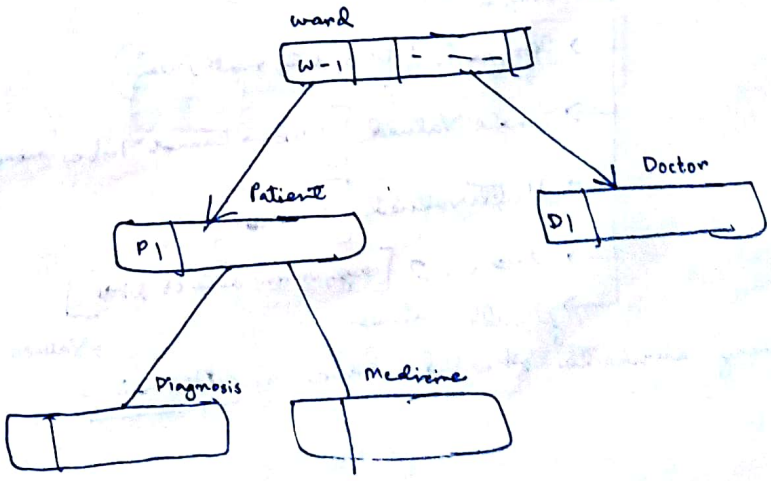
user_id	Book_id	Borrow date

- Conversion of E-R Model to Relational Model

3) Network Model:



3) Hierarchical Model:



Superkey: Set of attributes that can uniquely identify a record.

Candidate Key:

Minimal superkey

↓

cannot remove anything more ~~than~~ such that it still remains a superkey.

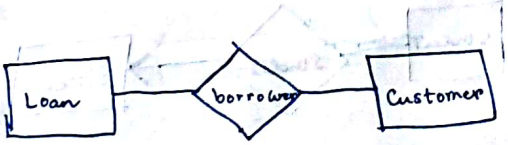
Primary Key: User defined candidate key.

Constraints

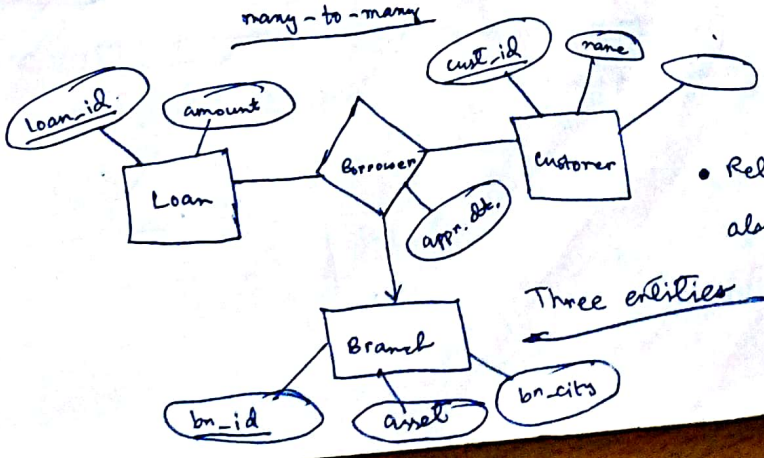
1) Cardinality
   - one-to-one
   - one-to-many
   - many-to-one
   - many-to-many

restricted to 2 / n

many-to-one



many-to-many

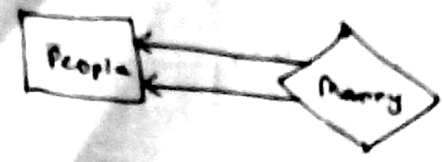


- Relation might also have attributes

Three entities

Borrower (br_id, loan_id, cust_id, appr_[redacted])

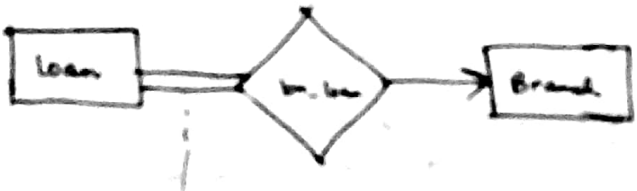
Degree of Relationship → Number of entities operated by a relation



Unary Relation

Unary Relation

Participation Constraints



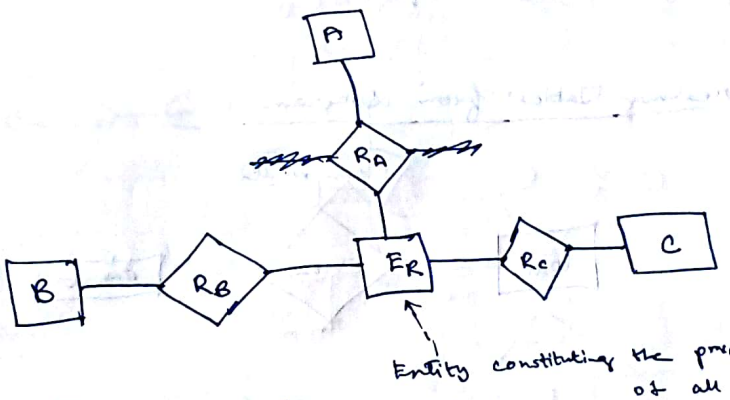
Total participation

Binary vs Ternary Relation

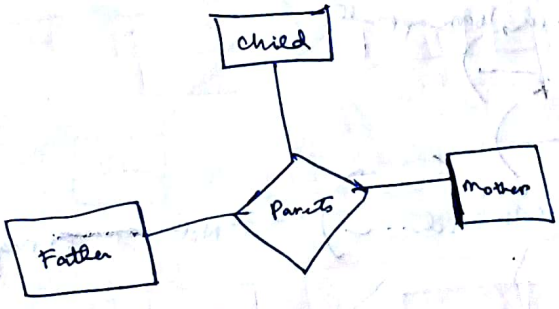
$E_1, E_2, \ldots E_n \rightarrow R$

$Prim(R) = Pri(E_1) \cup Pri(E_2) \ldots \cup Pri(E_n)$

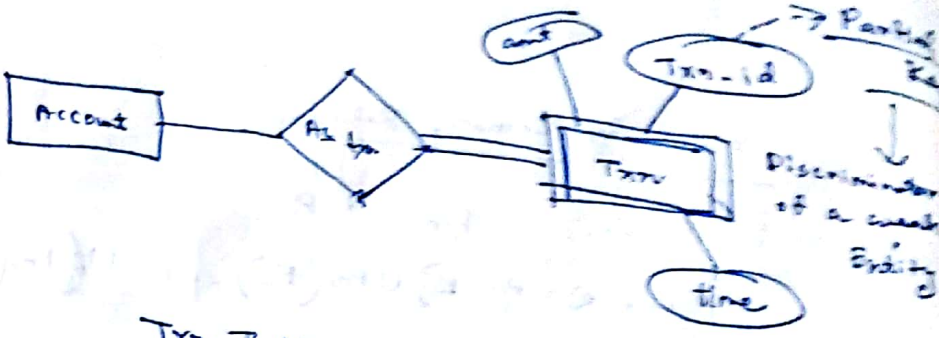
ternary



Entity constituting the primary keys of all the entities.



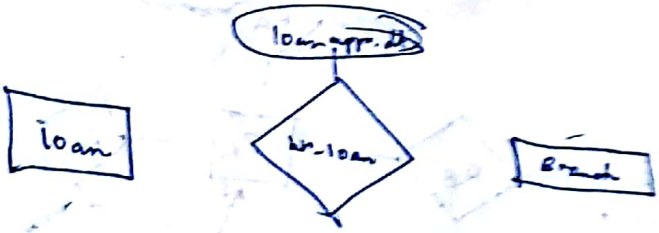
- Entities which have a primary key → Strong Entity
- Entities which do not have a primary key → Weak Entity.



Txn Table

Acc No	txn_id	--	--

Creating Tables from Diagram



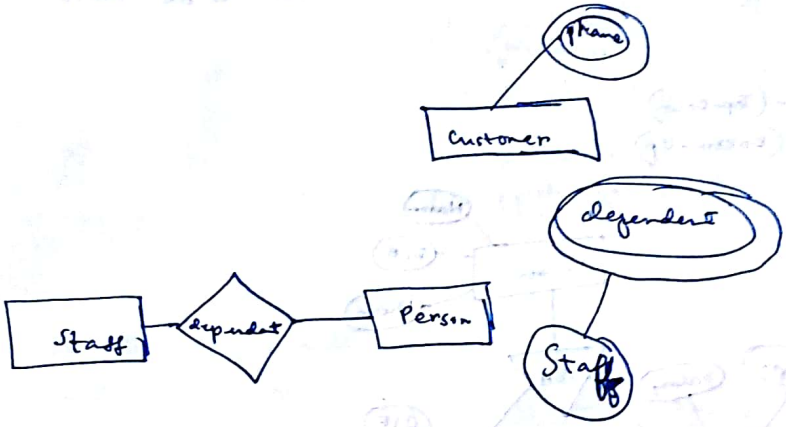
loan (loan_id, br_id, loan_appr_dt)

branch (br_id, ———)

br_loan (br_id, loan_id ...) --→ Not required

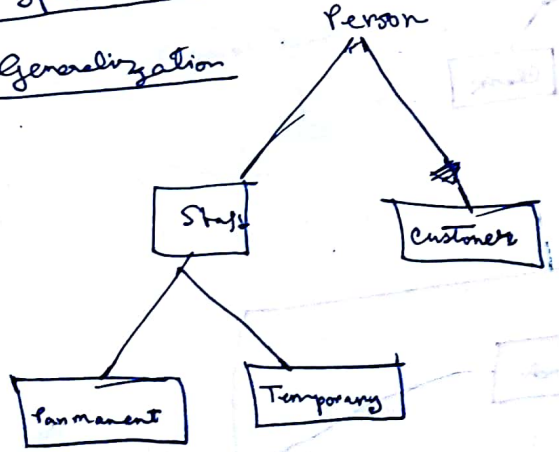
- We create two Tables, from 3 entities.

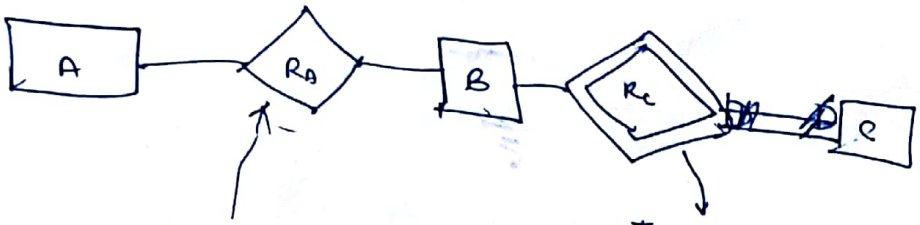
Customer



Specialization

Generalization

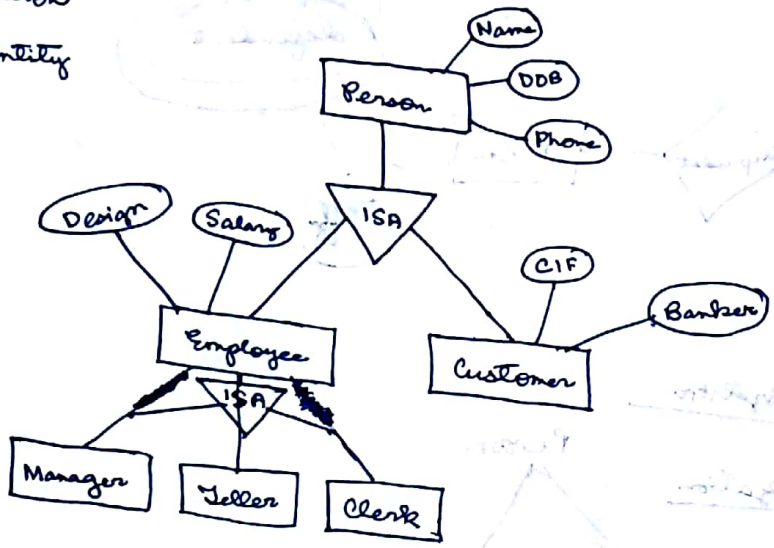




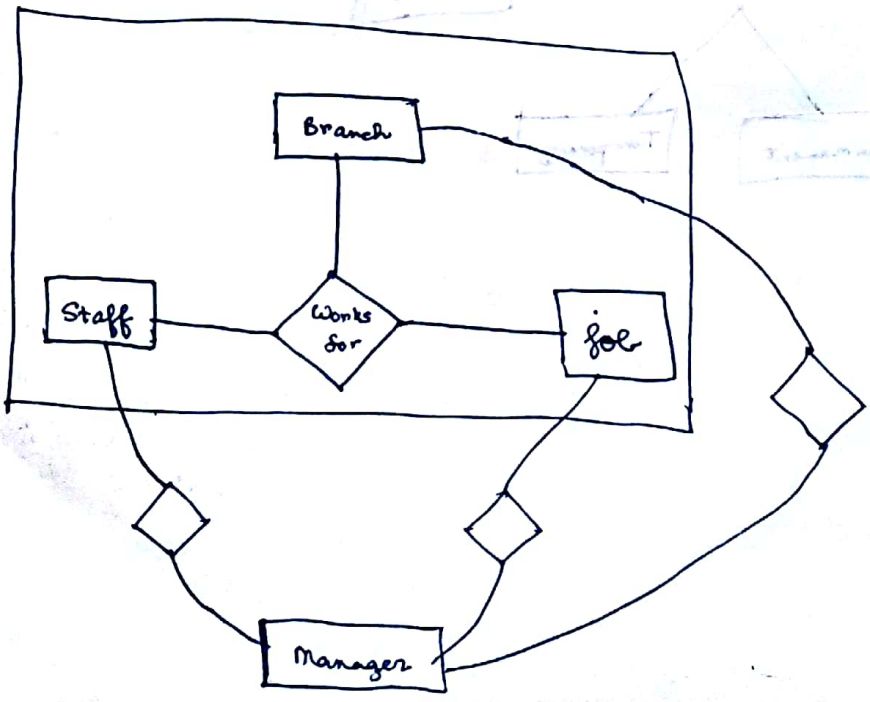
a table needs to be found

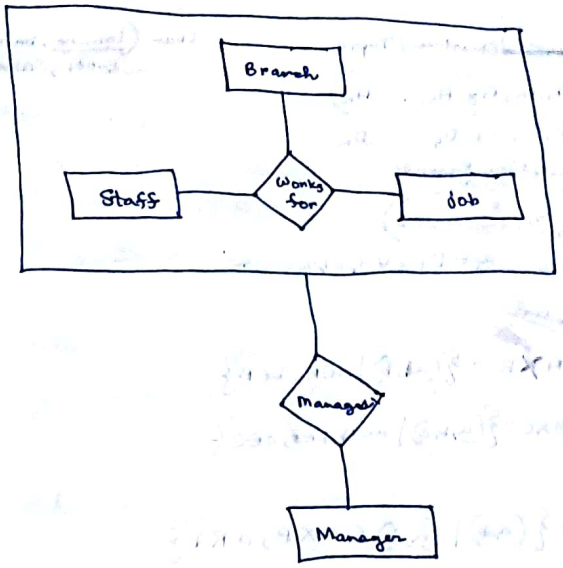
In case of weak entity, R_C does not need to be table.

- Specialization (Top-Down)
- Generalisation (Bottom-Up)
- Aggregation
- Weak Entity



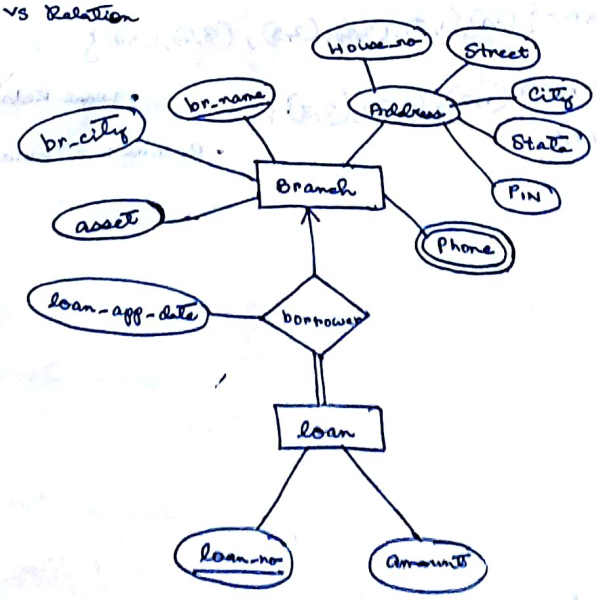
Aggregation





— Attribute vs Attribute

— Entity vs Relation



• How to store in Table?
Directly store House_no, Street,.... and don't store Address

Relation / Structure / Tuple

$R(A_1, A_2, A_3, A_4, \ldots, A_n)$

$D_1\ D_2\ D_3\ D_4 \quad D_n$

Relation Records

$r(a_1, a_2, \ldots, a_n)$

$a_i \in D_i, \forall i = 1, 2, \ldots n$

loan (loan_no, br_name, amount)

L-001, Gaala, 1000000

Cartesian Product

$A \times B = \{(a,b) \mid a \in A, b \in B\}$

$A \times B \times C = \{(a,b,c) \mid a \in A, b \in B, c \in C\}$

$A\ R\ B = \{(a,b) \mid (a,b) \in A \times B, a R b\}$

$A = \{1,2,3\} \quad B = \{2,3\}$

$A \times B = \{(1,2), (1,3), (2,2), (2,3), (3,2), (3,3)\}$

$A\ R\ B = \{(1,2), (1,3), (2,3)\}$

$R = \langle \ldots \rangle$

- Legal Relation
- Record in a Table → Relation

SQL / Relational Algebra

> create table branch
(br_name varchar(20),
br_city varchar(20) unique,
asset numeric(10,2) check(asset >= 10000000.00),
primary key (br_name));

branch (br_name, br_city, asset)
customer (cust_name, cust_st, cust_city)
loan (loan_no, br_name, amount)
borrower (cust_name, loan_no)
account (acc_no, br_name, balance)
depositor (cust_no, acc_no)

> insert into branch values
('Saltlake', 'Kolkata', '10000000');

- gender char(1) check gender in {'M', 'F'},

> create table loan
(loan_no varchar(10),
br_name varchar(20),
amount numeric(10,2),
primary key(loan_no),
foreign key br_name references branch(br_name));

- Foreign key: A single or a set of attributes whose of a table which serves as a candidate key for another table.

Update

update branch
set asset = asset * 1.1;

update branch
set br_city = 'Kolkata'
where br_name = 'Saltlake';

delete

delete from branch
where br_name = ' ';

delete from branch
where br_city = 'Kolkata';

SQL / Relational Algebra

branch (br_name, br_city, asset)
customer (cust_name, cust_st, cust_city)
loan (loan_no, br_name, amount)
borrower (cust_name, loan_no).
account (acc_no, br_name, balance)
depositor (cust_no, acc_no)

> create table branch
(br_name varchar(20),
br_city varchar(20) unique,
asset numeric(10,2) check(asset >= 10000000.00),
primary key (br_name));

> insert into branch values
('Saltlake', 'Kolkata', '10000000');

- gender char(1) check gender in {'M', 'F'},

- Foreign key: A single or a set of attributes of a table which serves as a candidate key for another table.

> create table loan
(loan_no varchar(10),
br_name varchar(20),
amount numeric(10,2);
primary key(loan_no),
foreign key br_name references branch (br_name));

Update

update branch
set asset = asset * 1.1;

update branch
set br_city = 'Kolkata'
where br_name = 'Saltlake';

Delete

delete from branch
where br_name = ' ';

delete from branch
where br_city = 'Kolkata';

- drop → Deletes the entire table / schema

Select

> select * from branch; → displays all the records

> select br_name, br_city from branch
  where br_city = 'Kolkata';

> select br_name, br_city from branch
  where asset > 10 cr AND (br_city = 'Kolkata' OR
  br_city = 'Mumbai');

> select br_name, br_city from branch
  where asset > 10 cr AND br_city in {'Kolkata', 'Mumbai',
  'Chennai', 'Delhi'};

- To Sort in order → order by br_city
- To Sort in Descending Order → order by br_city desc;

Functions (Aggregate)

- Average (avg)
- Minimum (min)
- Maximum (max)
- Total (sum)
- Count (count)

> select count(*) from branch; ⟶ Number of records in the table

> select avg(asset) from branch; ⟶ Avg. asset of all the branches

> select br_city, avg(asset) from branch; ⟶ Avg asset of each city

```
> select br_city, avg(asset)
    from branch
    where br_city = 'Kolkata';
```

```
> select br_city, avg(asset)
    from branch
    order by br_city
    group by (br_city);
```

```
> select br_name, sum(amount)
    from loan
    group by (br_name);
```

• View

Joining Tables

```
> select br_city, sum(amount)
    from branch, loan
    where loan.br_name = branch.br_name
    group by(br_city);
```

! ⟶ This attribute should be selected and should not be present as an argument inside the aggregate function

```
> select br_city, loan.br_name, loan_no, amount
    from branch, loan
    where branch.br_name = loan.br_name;
```

- customer_names who have borrowed a loan.

> select cust_name from borrower;

- To stop something from appearing multiple times

> select distinct cust_name
from borrower.

Select those cust_names which have both an account and have taken a loan.

> select cust_name
from borrower
in (select distinct cust_name
from depositor);

customers in borrower but not in deposit

> select cust_name
from borrower
not in (select distinct cust_name
from depositor);

Other way:

> select borrower.cust_name
from borrower, depositor
where borrower.cust_name = depositor.cust_name

Relational Algebra

branch (br_name, br_city, asset)
customer (cust_name, cust_city, phone)
loan (loan_no, br_name, amount)
borrower (cust_name, loan_no)
account (account_no, br_name, balance)
depositor (cust_name, account_no)

- SQL → Non-Procedural Language

$R(A_1, A_2, \ldots A_n)$

$A_i \in D_i$

$r(a_1, a_2, \ldots, a_n)$

Operations in Relational Algebra:

1) σ (Select) :---> Unary Operation

Notation: $\sigma_p(E)$ ← relation
↑ Predicate / Conditions

In SQL:
```
select * from branch
where br_city = 'Kolkata';
```

```
select * from branch;
```

In Relational Algebra: $\sigma_{br\_city='Kolkata'}(branch)$

$\sigma(branch)$

$\sigma_{asset>10000000}(branch)$

2) Π (Projection) :---> Unary Operation

SQL:
```
select br_name, asset from branch
where br_city = 'Kolkata';
```

$\Pi_{br\_name, asset}(branch)$ → Displays all br_names and asset of all relations.

$\Pi_{br\_name, asset}(\sigma_{br\_city='Kolkata'}(branch))$

$\sigma_{br\_city='Kolkata'}(\Pi_{br\_name, asset}(branch))$ ✗ Not Possible

$\sigma_{asset>10cr}\left(\Pi_{br\_name, asset}(branch)\right)$ → Possible;
Display br_name and ~~city~~ asset.

3) RUS (Union): → Binary Operation

SQL:

```
select cust_name
from customer, ~~loan~~ borrower, depositor
where borrower.cust_name = customer.cust_name
      ~~AND~~ OR depositor.cust_name = customer.cust_name;
```

OR

```
select cust_name from borrower
union
select cust_name from depositor;
```

Relational Algebra:

$\Pi_{cust\_name}(borrower) \cup \Pi_{cust\_name}(depositor)$

No. of attributes, all their domain should be same

$\Pi_{cust\_name, loan\_no}(borrower) \cup \Pi_{account}(depositor)$ ✗ → Not Possible

- R and S should have same arity → means no. of attributes should be same
- Domains of the attributes should be same

4) ~~Set~~ R-S (Set Difference):

SQL:

```
select cust_name from borrower
except
select cust_name from depositor.
```

- R and S should have same arity.
- Domain of the attributes should be same.

$A = \Pi_{cust\_name, loan\_no}(borrower)$

$B = \Pi_{cust\_name}(borrower) - \Pi_{cust\_name}(depositor)$

$\sigma(A \bowtie B)$

$A \bowtie$

5) $\rho_x(E)$ (Rename):

↓ new name

- $\rho_x(\Pi_{cust\_name}(B) - \Pi_{cust\_name}(D))$

$\sigma_{cust\_name\ in\ x}(borrower)$

Rename E as X:

$\rho_x(E)$ → Relation Name

6) Cartesian Product:

r(A, B)

α	1
β	2

S(C D E)

α	10	a
β	10	a
β	20	b
γ	10	b

r × s

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

- Find the largest account balance in the bank.

ac_no br_name balance d.acc_no d.br_name d.balance

$$r_1 = \Pi_{\text{account.balance}} \left(\sigma_{\text{account.balance} < \text{d.balance}} \left(\text{account} \times \rho_{\text{d}}(\text{account}) \right) \right)$$

$$r_2 = \Pi_{\text{balance}} (\text{account})$$

$$r_2 - r_1 \quad \text{(Ans)}$$

- Find all customers who live on the same street and in the same city as 'Ram'.

$$\Pi_{\text{customer.cust\_name, customer.cust\_street, customer.cust\_city}} \left(\sigma_{\text{street} = \text{d.street} \wedge \text{d.cust\_name} = \text{'Ram'} \wedge \text{city} = \text{d.city}} \left(\text{customer} \times \rho_{\text{d}}(\text{customer}) \right) \right)$$

① $\rho_{\text{d}}(\sigma_{\text{name} = \text{Ram}}(\text{customer}))$

↓

'Ram's record will also be included.

To leave it out do Set Difference.

② $\rho_{\text{d}}(\Pi_{\text{street, city}}(\sigma_{\text{name} = \text{Ram}}(\text{customer})))$

- Example:

r (A B)

A	B
α	1
α	2
α	3
β	1
γ	1
δ	1
δ	3
∈	G
∈	1
β	2

S (B)

B
1
2

r ÷ s : (A)

A
α
β

r ÷ s Structure

(R) (S) r ÷ s → (R − S)

A Tuple t is in r ÷ s iff —

1) t is in $\Pi_{R-S}(r)$

2) For every tuple t_s in s, there is a tuple t_r in r satisfying both the followings :

   a) $t_r[S] = t_s[S]$

   b) $t_r[R-S] = t$

loan_no	br_name	amount
L-170	Saltlake	10L
L-230	Garia	20L
L-260	Esplanade	30 L

Borrower

cust_name	loan_no
Ram	L-170
Jadu	L-230
Madhu	L-465

Functional dependency and Normalization

Relational database

Requirements

A = { br_name, br_city, asset, cust_id, cust_name, cust_street, cust_city, loan_no, amount, account_no, balance }

branch (br_name, br_city, asset)

customer (cust_id, cust_name, cust_street, cust_city)

loan (loan_no, br_name, amount, cust_name)

account (account_no, br_name, balance, cust_name)

- Redundancy
- Modification anomaly
- Insertion ~~anomaly~~ anomaly
- Deletion anomaly

Dependencies:

br_name → br_city

br_name → asset

cust_id → cust_name

loan_no → amount

- br_name determines br_city
  OR
  br_city depends on br_name

Functional dependency: $A, B \subseteq R$ A functionally determines B,

or B is functionally dependent on A,

determinant ← A → B if ~~∀~~ tuples $t_1 \times t_2$ in ~~∀~~ ? (R)

when $t_1(A) = t_2(A)$ then $t_1(B) = t_2(B)$

A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_2	c_2

a	b	c
a_1	b_1	c_1
$a_{\neq 1}$	b_2	c_2
a_2	b_2	

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_2	c_2

a	b	c
a_1	b_1	c_1
a_2	b_2	c_2

→ Join to get back the previous table

↓

a_2 b_2 c_1

↓

one record extra

- Not Desirable

Reflexivity (Trivial Dependency)

br_name → br_name

~~br_name → br_city~~

br_name, br_city → br_city

br_name, asset → br_city, asset (Augmentation)

- Any set of attributes can determine its subsets

br_name → br_city

br_city → br_state, then br_name → br_state (Transitivity)

R

X, Y ⊆ R

Armstrong's Axioms

1. X → Y is trivial if Y ⊆ X (Reflexivity)
2. X → Y, then XZ → YZ | (Z ⊆ R) (Augmentation)
3. X → Y, Y → Z, then X → Z (Transitivity)

Secondary Rules

1) If X → Y and X → Z, then X → YZ (Union Rule)
2) If X → YZ, then X → Y and X → Z (Decomposition Rule)
3) If X → Y and WY → Z, WX → Z (Pseudo Transitivity)

Ex: R(A, B, C, D, E, F)

F = {A → BC, B → E, CD → EF}

Show that AD → F.

~~∴ A → BC,~~

F' {A → B, A → C, B → E, A → E, A → BC

CD → E, CD → F, CD → EF}

- F and F' are equivalent

A → C,

CD → F

∴ AD → F (By Pseudo Transitivity)

F & D cover

- F' is a cover of F.
- F is a cover of F'

One satisfies means other also satisfies.

- We will be interested in finding the minimal cover.

Example:

$R(A, B, C, D)$

$F = \{A \to BC,\ B \to C,\ A \to B,\ AB \to C,\ AC \to D\}$

Step I:

$= \{A \to B,\ A \to C,\ B \to C,\ A \to B,\ AB \to C,\ AC \to D\}$ [make attributes on the right side a single attribute]

Step II: Make attributes

$AC \to D$

$A \to C$

$AC \to C$

$A \to AC$

$A \to D$

Also, $AB \to C$ can be written as $B \to C$

$= \{A \to B,\ A \to C,\ B \to C,\ A \to B,\ B \to C,\ A \to D\}$

$= \{A \to B,\ A \to C,\ A \to D\}$

- Minimal cover may not be unique.

$= \{A \to B,\ B \to C,\ A \to D\}$

Minimal cover // Canonical Cover

X → Set of attributes

X^+ → closure of X

$X \subseteq R$

$R(A, B, C, D)$

$F = \{A \to BC, B \to C, A \to B, AB \to C, AC \to D\}$

$\{A\}^+$ $\overset{A \to A}{\curvearrowright} X = \{A\}$

- Find dependencies of the form $\alpha \to \beta$ where $\alpha \subseteq X$.
- If possible to find, change $X = X \cup \beta$

$A \to BC$

$X = \{A, B, C\}$

$AC \to D$

$X = \{A, B, C, D\}$ ← closure of A $(\{A\}^+)$

∴ A is a superkey as its closure contains all the attributes. (i.e. $X = R$)

$\{B\}^+$ $\quad B \to B$

$X = \{B\}$

$B \to C$

$X = \{B, C\}$ → closure of B $(\{B\}^+)$

Candidate Key

1) $X^+ = R$
   $X \to$ is a superkey

2) $\exists$ no $Y (\subset X)$ and $Y^+ = R$

1. Find the candidate keys—

R (A, B, C, D)

F = { AB → C , AB → D , C → A, D → B }

$\{C\}^+$ C → C

X = { C }

C → A

X = { C, A }

$\{D\}^+$ D → D

X = { D }

D → B

X = { D, B }

$\{AB\}^+$ AB → AB

X = { A, B }

AB → C

X = { A, B, C }

X = { A, B, C, D }

$\{AD\}^+$

X = { A, D, B, C }

$\{CD\}^+$

X = { C, D, A, B }

$\{BC\}^+$

$\{AB\}^+ = A^+ \cup B^+$ ✗ Not Necessarily True

If $A^+ \cup B^+ = R$ then AB is a superkey.

$AB^+ = R$