SQL

Structured Query Language

Data Modeling —

↳ Topic Modeling

↳ Context Oriented

↳ History || Store & Retrieve the data

↳ Query

} OpenAI
Deepseek

Customers

#	name	place	parent_id
1	Bob	Blr	null
2	Bob	Che	1
3	Bob	Del	2
4	Bob	Blr	null
5	Bob	Pune	3
6	Bob	Che	4
7	Bob	Blr	6

Search with name

Customers

id	name	place	assign
1	Bob	Blr	1
2	Bob	Blr	2
3	Bob	Che	1
4	Bob	Hyde	1
5	Bob	Del	2
6	Bob	Che	2

Type 3

#	name	place
1	Alex	Blr
2	Aman	Che
3	Sachin	Hyderabad

Python
Place Key
: 1 &
: Place = Pune

Dynamically add
class

Fact Table [Measurable.

Factless Table [Not measurable | Not deep analysis]

Measures - Additive, Semi-Additive, &

Non-additive → % of interest rate

Sum of deposits

Sum of balances across customers but over time

Student#	Course#	EnrollmentDate
1	5	—
2	6	—
3	7	—

facts

Which Student enrolled in which course?

fact Tables

- Transactional
- Periodic
- Accumulative

Relationships

- 1 to 1
- 1 to M or M to 1
- M to M
- Weak Relation -

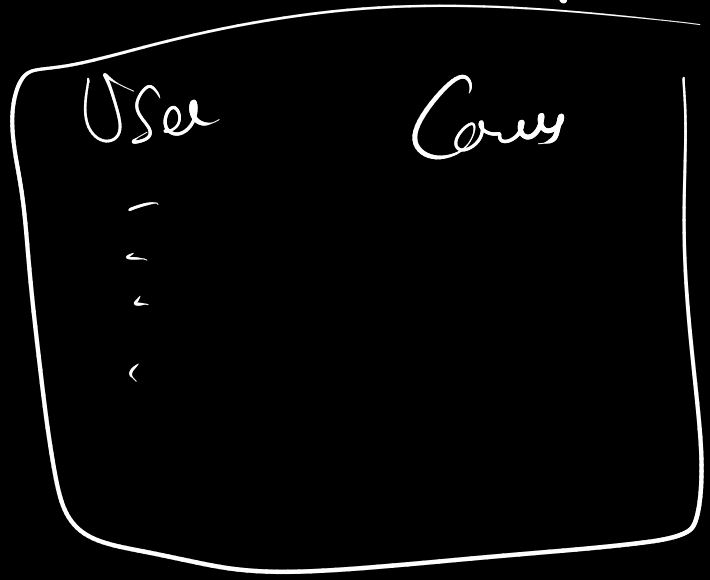
One to one

User — Addhaar Card

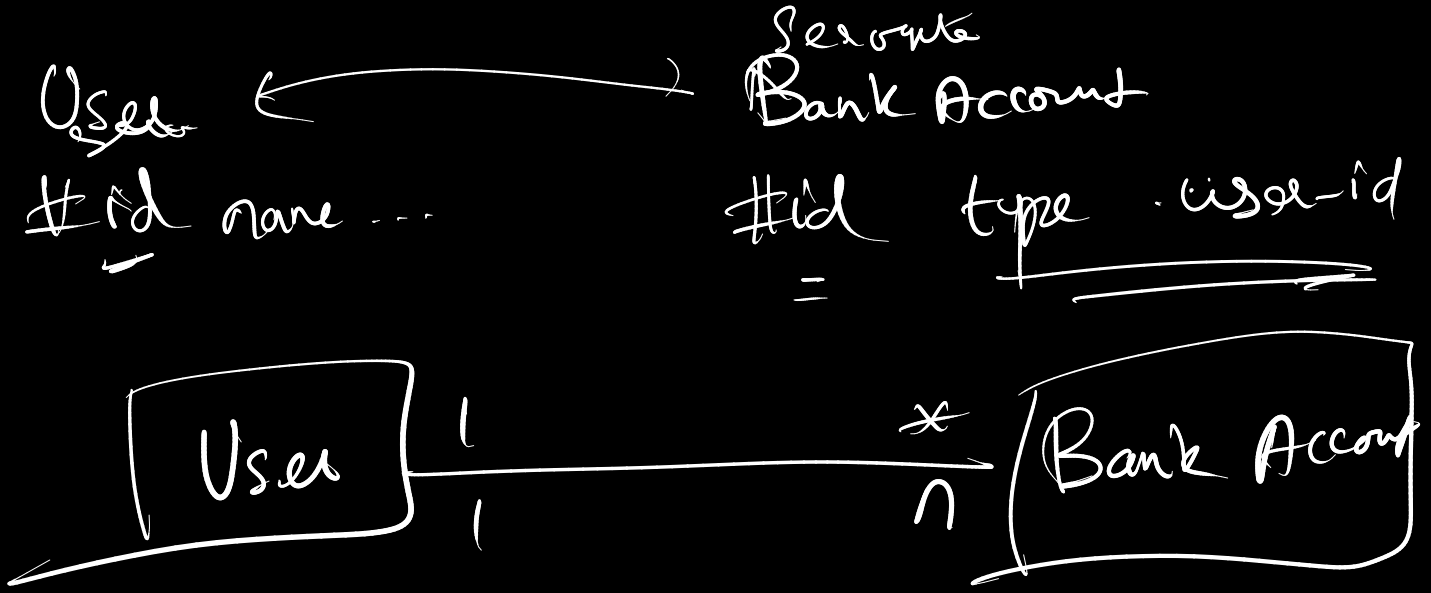
User

name address mobile email aadhaar no

One to Many Relation.



Teacher


Secondo
User
Bank Account
#id name ...
#id type . user-id
=
User
1
1
*
n
Bank Account

User

id	name
1	Amy
2	Alex
3	Bob
4	
⋮	⋮

Account

id	AccountType	User_Id
1		1
2		2
3		2
4		2
5		3
⋮		⋮

1. Join
2. on + Join
3. SubQueries

```
Select * from user Join
    Account on account.user_id = user.id
    where user.name like = " ";
```

↳ Schemas
Table } Handlers

A thread opens ↳ Query —, RAM

closed,

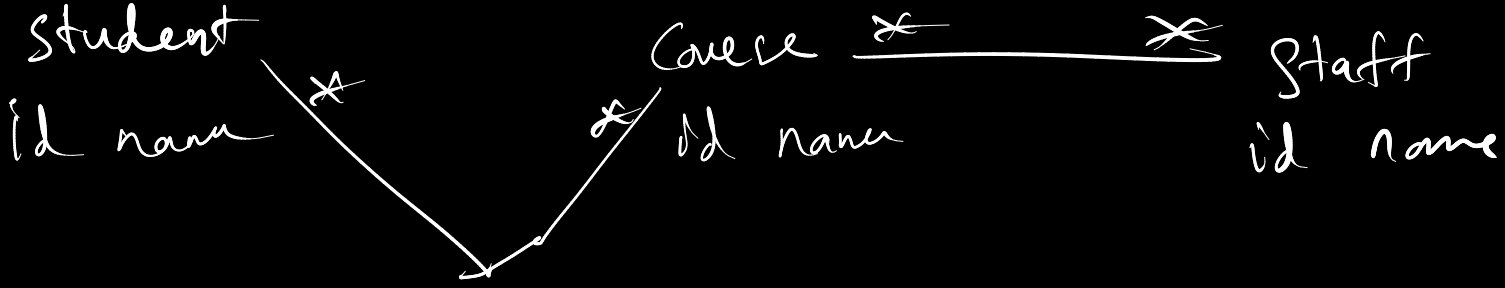
Many to Many Relationship

Student

Course

Student_course

student_id course_id


Student
id name
Course
id name
Staff
id name

MySQL Architecture

- Client Layer
  - Connections & Communication
- SQL Layer
  - Query Processing & Optimization
- Storage Layer
  - Data Storage & Retrieval

Client Layer [Conn & Comm]

1. Clients
   - command
     - mysql -u username -p
   - phpmyadmin / MySQL Workbench.
   - JDBC, Python connectors (Drivers) & Alchemy

1) Connection Handling
   - Thread based
2) Connection Pools
   - Threads
     [Min & Max] threads
   - Each Client will have a separate connection threads.

- Manages Authentication.
  - Threads, pools
  - Security.

Transfers data Query & output

SQL Layer

↳ Query Parser ⟶ Syntaxes (tokens in string Syntax error)

↳ Query Optimizer ↳ Index, plans table name, Attribute name

↳ Query Execution Engine

↳ Step by step.

retrieves the data from Storage Engine

Storage Engine.

InnoDB [ACID]

- Transactions,
- Foreign keys
- Row Level locking

MyISAM

- Reads fast
- Table Level Locking
- X foreign key
- Search Engines

Heap [Memory]

↳ Stores in RAM

↳ Non Persistent

CSV

Archives – Log Data.

NDB [Cluster] – High Availability & fault Tolerance

How MySQL Stores Data in the hard drive

/var/lib/mysql/<db_name>

ls /var/lib/mysql/sigmoid

table_name.frm → Stores table definition

table_name.ibd → Table data & indexes (InnoDB)

table_name.MYD → Table data [MyISAM]

table_name.MYI → Index (MyISAM)

Buffer Pool
(Cache Mechanism)

↳ INNODB - Buffer Pool → Performance Boosting

Transaction Mechanism

1) Data Integrity

1) ACID

Atomicity → All or nothing Executed

Consistency → Maintains the State

Isolation → No Interference

Durability → Commits / Save Permanently.

Atomicity

ATM

↳ withdraw 2K

Bank deducts from your account

↳ System fails.

↳ Roll back

Consistency.

ATM

10k 9.5k Balance.

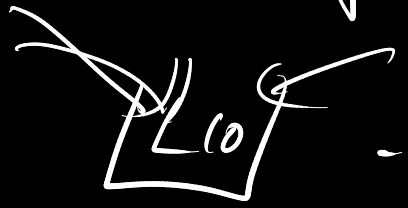
↳ State

Online Purchase - Cross

Isolation.

Eg: Book the ticket

flight / Train / Bus



Locked up

Durability (Permanently stored)

↳ 10k → ATM

Correct } Stats

Crashes

Data Normalization

- Easy Storage (Memory Efficient)
- Faster Access

1 Nf [first Normal form]

Rules

- No repeating groups
- Atomic values [Indivisible values]
- Unique Columns
- Identify a prime key.

Order_id	Cust_name	Items_ordered	Quantity
1	Alice	Pizza, Beer	4, 1
2	Bob	Sandwich, Coke	2, 3
3	Dan	Pasta	4, 6
4	Amar	Breads, Dosa	2, 3

Order id	Cust name	Item ordered	Quantity
1	-	Pizza	2
1	-	Burger	3
2	-	-	-
2	-	-	-
3	-	-	-

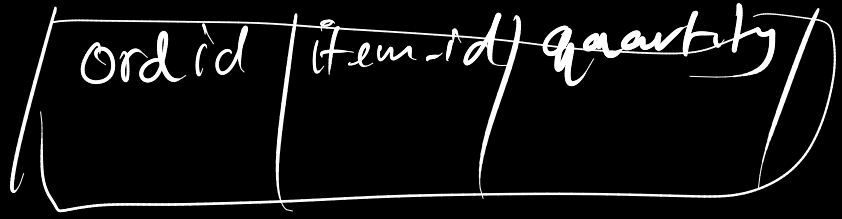
Increased Data Redundancy.

Slower Performance.

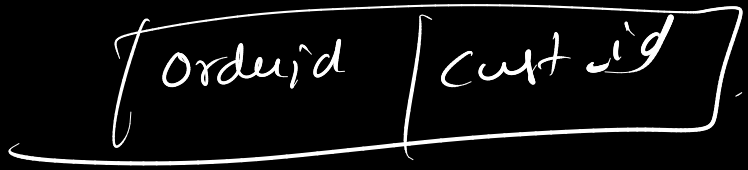
2NF - Partial Dependency Issues

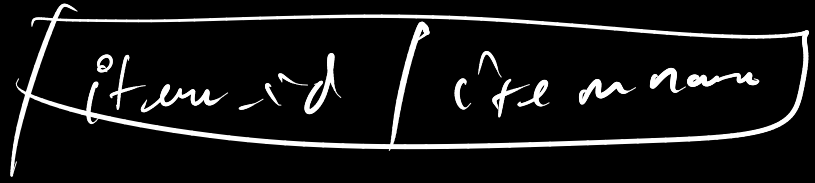
- 1NF
- No partial Dependency
  - Every non key column must dependent on the primary key

order_id	cust_id	cust_name	item_id	item_name	Qua
1	C01	Alice	101, 102	Pizza, Burg	2,3
2	C02	Bob	104, 106	Sand, Coke	2,2
3	C03	Aman	105, 106	Pasta, Lemo	2,2


Ord id
item_id
quantity


cust_id
cust no


order_id
cust_id


item_id
item name

- More tables, more joins
- Increased Data complexity
- More foreign keys

3 Nf

↳ 2 Nf → 1 Nf

↳ No Transitive Dependency :

↳ A non key column should not depend on another non-key column

↳ Every non key attribute should depend only on the primary key.

ord_id	cus_id	cus_name	item_id	item_name	city	zipcode
1	1	Alex	100,110	Idly, Dosa	Blr	60001
2	3	Bob	200,201	Bond, Coffee	Chen	60002
3	10	Jmar	300,301	Noodles, Coke	Hyd	60005
4	12	Amit	301, 302	Coke, Chapa Rotis	Pune	6008 6

→ 1) Many Tables
2) Join Query

Slow

More Complexity

4 NF – Multi-Valued Dependency Issue

Teacher_id	Teacher_name	Subject	Classroom
100	Alex	Math	10
100	Alex	Math	11
101	Bob	Science	10
101	Bob	Science	11
102	Adam	English	12
102	Adam	English	13

MySQL DDL

- Create
- Alter
- Drop
- Truncate
- Create Index

Create Constraints

Create Table with Constraints

Department
id
name

Employee
id
name
dept_id
Salary
email

Indexing in MYSQL

Create table employee
id (Primary key)
name
email (unique)
-

Select * from employa where email like "%_%";

- primary key

  Select * from table where id=

- unique

  Select * from table where email like ' '

If name

create index idx_name on employee(name)

Select * from employee name like "Arun"

Regular Index

Composite Index

Create INDEX idx_name_dept
(user defined)
on employee(name, department)

Unique Index:

- Unique keyword (No duplication)
- Full Text Index for fast Text Searching

Create FULLTEXT INDEX index_bio
on employee (bio);

Select * from employee where

MATCH (bio) AGAINST ('Python Developer');

↓ column name

faster than.

Select * from employee where bio like '% Python Developer %';

When Should I use Indexe?

- frequent search.
- sorting result (Order by)
- Joining tables.

Limitations.

- Slow down Insert, Update, Delete ops
- Consumes extra storage.

Remove an Index

Drop INDEX ind_name on employee

Select * from employee where

MATCH (name) AGAINST (' ' IN

NATURAL LANGAGE

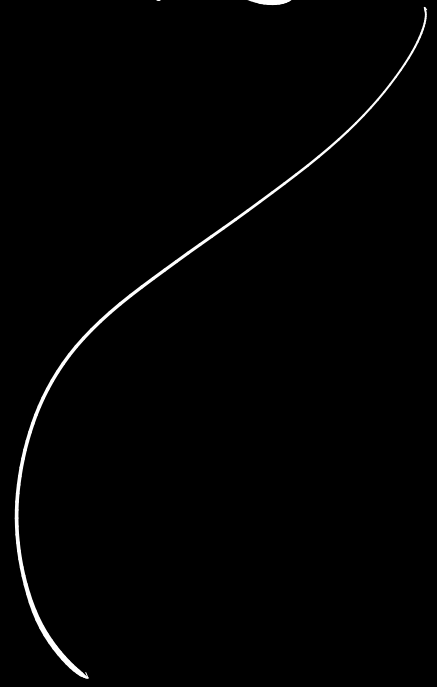
MODE)

Select * from employee where
MATCH(name) AGAINST ('+Amal -Nath'
IN BOOLEAN MODE)

EXPLAIN

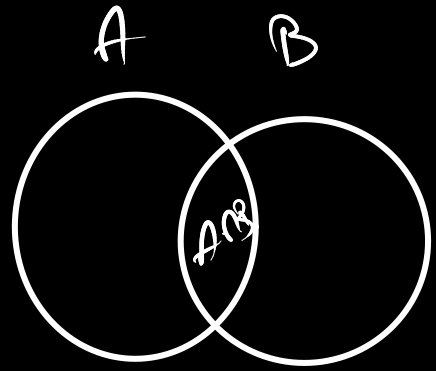
DML (Data Manipulation Language)

- Insert
- Update
- Delete
- Select



SQL Join

- INNER Join
- Left Join
- Right Join
- full Join
- Self Join



A∩B

A

B

A∪B

Employee

id	name	dept	salary
1	Alice	10	5k
2	Bob	20	6k
3	Charlie	10	7k
4	David	30	8k

Dept

id	name
10	HR
20	IT
30	fin
40	Mark

Filtering & Ordering Data

- where clause
  - AND [Repeatedly]
  - OR = ( ) Parathesis for subquery

Sorting & Limit

Key

ORDER BY

LIMIL

Select * from
— where — —
order by id ASC
or
DESC

LIMIL 100

Correlated subqueries.

```
Select * from emp e1 where
    Salary > ( Select avg(salary) from
        emp e2 where e1.department =
                                e2.deptno)
```

Exp & Comparison

=, !=, <, >, <=, >=,

BETWEEN, LIKE, IN.

Select * from product where price between 100 AND 500 Order by name DESC limit 10

select * from prod where itemid
IN (select id from item - -);

↓

List of ids

GROUP & AVG Clause

```
Select department, AVG(salary) As avg_sal
from employee group by department
HAVING AVG(salary) > 10k ;
```

Regular Expression

Select * from customer where name
REGEXP '^A|B'; → PIPE Symbol

Ex :- find all regular expression formats

CTE [Common Table Expressions]

```
with highsalary AS (
    select name, salary from employee
    where salary > 10k
)
select * from highsalary;
```

temporary table highsalary

Hierarchical Queries

```
Select * from employee
    START WITH manager_id IS NULL
    CONNECT BY PRIOR employee_id=manager_id
```

UNION, INTERSECT, EXCEPT (Set)

```
Select name from old_customer
UNION Select name from new_customer;
```

```
Select name from old_customer
UNION ALL Select name from new_customer;
```

INTERSECT

```
Select name from oldcustomers
INTERSECT
Select name from newcustomers
```

EXCEPT

```
select name from oldcustomers
EXCEPT
Select name from newcustomers
```

Creating views

CREATE VIEW high_volume AS

Select id, cust_id, total_amount

from order where total_amt > 100

high_volume = Virtual table

PROCEDURE → function or method

```
DELIMITER $$
Create PROCEDURE frequent_customer (IN minorder INT)
BEGIN
    Select cust_id, COUNT(order_id) as ord_count
    from order GROUP BY cust_id,
    HAVING order_count >= minorder;
END $$
DELIMITER ;

// CALL frequent_customer(10);
```

RANK, DENSE RANK, ROW_NUMBER.

```
Select * RANK() OVER (PARTITION BY department
  ORDER BY salary desc) AS rank,
 DENSE_RANK() Over (PARTITION BY department
  ORDER By salary desc) AS dense_rank
 ROW_NUMBER() over (PARTITION BY department
  ORDER BY salary DESC) AS row_number
  from employee ;
```

JSON Data

```
insert into customer (name, details)
values ('Alice', '{"age":30, "city":"blr"}'),
       ('Bob', '{"age":28, "city":"hyd"}');

select name, details ->> '$.city' as city,
             details ->> '$.age' as age
from customer;
```

PARTITION

Query optimization

```
Create table Sales ( id int not null,
    Sale_dat DATE NOT NULL,
    amount DECIMAL(10,2) NOT NULL,
    PRIMARY KEY (id)
) PARTITION BY RANGE (YEAR(Sale_dat))(
    PARTITION P2023 Values Less than (2024),
    PARTITION P2024 Values Less than (2025)
);
```

ROLL BACK

```
START TRANSACTION
    update account set balance = balance - 1000
    where account_name = 'Alice';
    update accun set balance = balance + 1000
        where account_name = 'Bob';
ROLLBACK!
Select * from Accounr;
```

SAVE POINT

```
START TRANSACTION;
update account SET balance = balance - 500 where
  account_name = 'Alice';
SAVEPOINT step1;
update account SET balance = balance + 500 where
  account_name = 'Bob'
SAVEPOINT step2;
update account SET balance = balance + 300 where
                                account_name = 'Bob'
ROLLBACK TO step2; COMMIT;
```

ROLLBACK in procedure

```
DELIMITER $$
Create procedure transfer_money (IN from_acc INT,
                                 IN to_acc INT, IN amount
                                 DECIMAL(10,2))
BEGIN
    DECLARE EXIT handler for exception
    BEGIN
        ROLLBACK;
        SIGNAL STATE '45000' SET MESSAGE_TEXT
                             = 'Transaction failed';
    END
```

```
START TRANSACTION;
update account set balance = balance - amount
where acc_id = from_acc;
update account set balance = balance + amount
where acc_id = to_acc;
COMMIT;
END $$
DELIMITER
```

Recursion

```
WITH RECCURSION employee_hira AS (
    SELECT * 1 AS level from employee
     where manager_id is NULL
    UNION ALL
    Sllect *, level+1 from employee
     INNER JOINT employee_hira ON eh.manager_id
                                = eh.emp_id
) Select * from employee_hira
```