Information :-

DBMS

#DBMS

Data :-

Anything who occupy space for storing. It's work as raw material for any processed information.

Information :-

After proven the data, the collective important and required data is known as information. This info. is way more helpful to reach to a conclusion or decission. (processed, organized, structured data)

⇒ Types of data :-

1) Quantitative :-
   This type of data is in numerical form,
   ex :- weight, volume, cost of an item.

2) Qualitative :-
   Descriptive, but not numerical.
   ex :- Name, gender, hair color of a person.

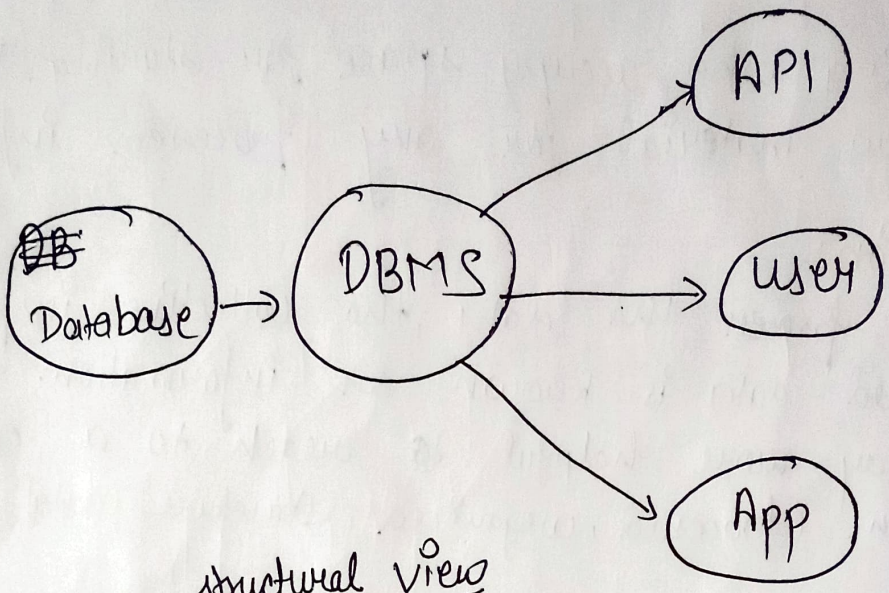
#DBMS

Database management system is a software or technology used to manage data from database.
Database is a system that enables you to store, modify and retrieve data in an organized way.
examples ⇒ MySQL, Oracle, MongoDB

⇒ Advantages :-

1) Redundancy of data and inconsistency is easy to handle.
2) accessing time of data is easy.
3) Integrity problem (easy to modify or add functionality)
4) Security is provided by DBMS.

5) Restricted access provided to users.



structural view

DBMS architecture :-

View of data :-
the representation of data visualization is different for different type of users, according to priority and needs.

Architecture :-
DBMS architecture help users to get their requests done while connecting to the Database. the architecture choosen depending on several factors like the size of the database, number of users, and relationship between users.
The main purpose of three level architecture is to enable multiple users to access the same data with a personalized view while storing the underlying data only once.

⇒ Types are :-

1) physical level / Internal level.
the lowest level of abstract, who describe how the data are stored, indexing, and organized on the disk or other storage device.

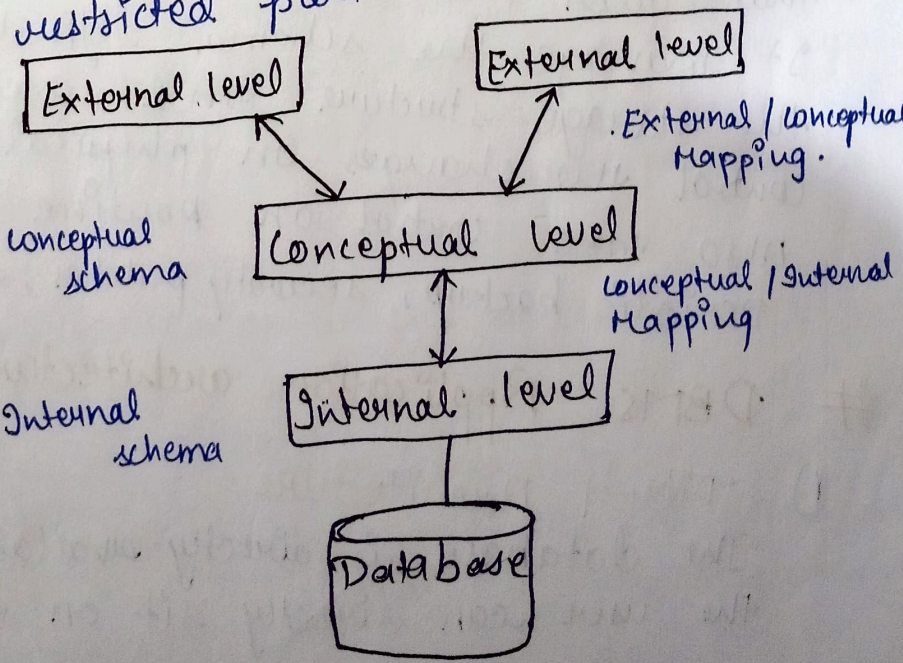
it has physical schema which describe physical storage structure of Database.

2) Conceptual level / Logical view.

It define the overall data structure, relationship, and constraints without detailing how data is physically stored.
Database administrators and designers work at this level to design the database schema.

3) View level / External level :-

This is the highest level of abstraction that directly interacts with end-users or applications.
It provide different view to different end-users.
Each view schema describes the database part that a particular user group is interested and hides the remaining database from that user group.
the database contains several schemas that sometimes called as subschema. The subschema is describe the different view of the database.
It also provide security to prevent users to access the restricted part.



Instances and Schemas

Collection of information stored in the DB at a particular moment is called an instance of Database. and schema is a structural description of data. schema doesn't change frequently. Data may change frequently. There types of schemas are: physical, logical, view schemas.

Data Models:-

It provide a way to describe the design of a Database at logical level.
It use a collection of tools for describing data, data relationships, data semantic & consistency constraints.
Ex:- ER model, Relational model, etc.

* for accesing the Database there is an interface for application. like Java Database connectivity (JDBC) in Java or open database connectivity (ODBC), in C/C++.

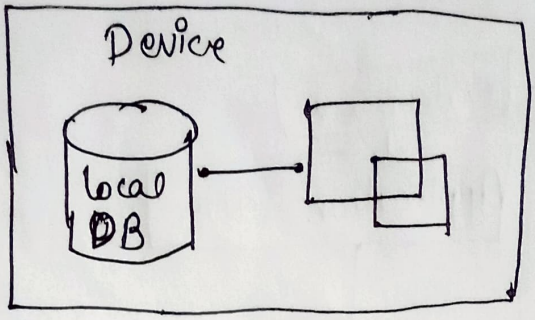
Database Administrator (DBA)

A Database Administrator is a person who has central control of both the data and the programs that accen those data.
It manage the schema defination. It manage the storage structure and accen methods.
control the changes on physical level modification
Also take control on routine maintenance. (periodic backup, security patches, and upgrade).

DBMS Application architecture:

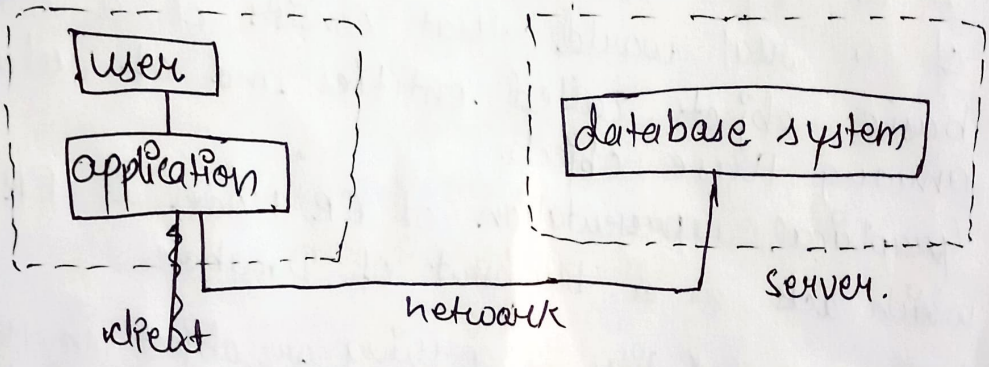
1) Tier-1 Architecture:-
The database is directly available to the user, the user can directly sit on the DBMS and

use it that is, the client, server, and Database are all present on the same machine.



2) Tier-2

The application at the client end directly communicates with the database on the server side. APIs like ODBC and JDBC are used for this communication. The application on the client side, ~~has~~ establishes a connection with the server side in order to communicate with the DBMS.



3) Tier-3

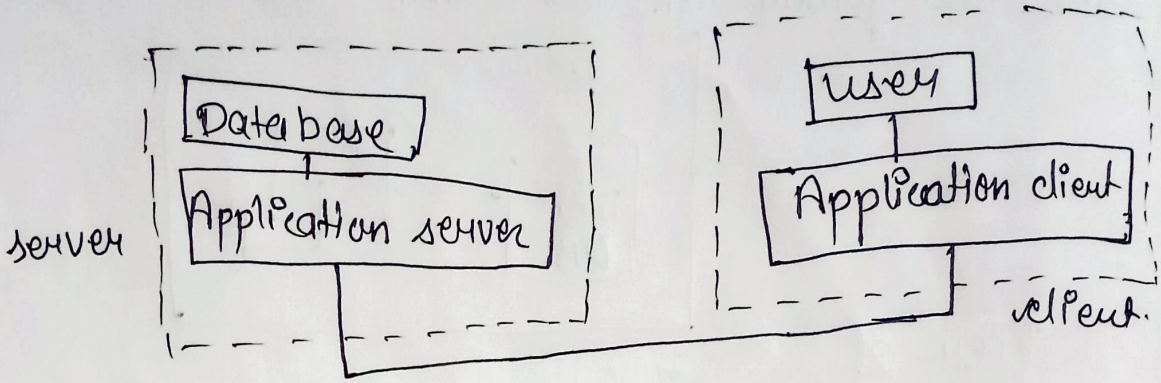
In Tier-3 client ~~can~~ does not directly communicates with the server. Instead, it interacts with an application server which further communicate with the database system and then the query processing and transaction management takes places.

This type of application an architecture work on large web server.

for this, it has a few advantages like:-

Data Integrity:- App server acts as a middle layer between client and DB, which minimize the chances of data corruption.

security : client can't directly access DB, hence it is made secure.

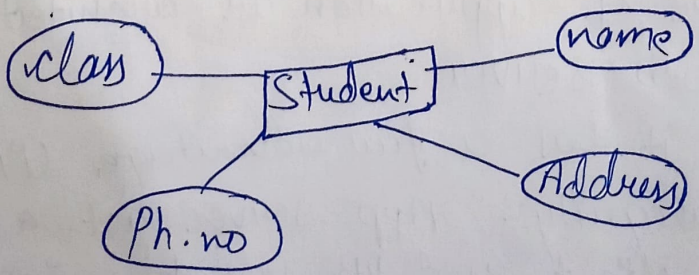


ER Model :-

Data model : collection of conceptual tools for describing data, ~~data~~ data relationship, data semantics, and consistency constraints.

It is high level data model based on a perception of a real world that consists of a collection of basic objects, called entities and of relationship among these objects.
Graphical representation of ER Model is ER diagram, which acts as a blueprint of Database.

Entity :- An Entity is a thing or object in the real world that is distinguishable from all other objects.
→ It has physical existence
→ Entity can be uniquely identify.
→ Strong Entity : can be uniquely identify.
→ weak Entity depends on strong entity for existence.



→ Entity set :-

The set of same number of entities, which share same attributes as properties.

Eg. student is an entity set (which share attributes like name, class, add. etc)

→ Attributes :-

Attributes is the property of entity, which define the Entity. Each entity is represented by a set of attributes. for each attribute, there is a set of permitted values, called the domain, or value set of that attributes.

ex:- student_ID, name, phone no.

Type of Attributes :-

1) Simple attributes.
Attribute which can't be divided further.
ex: account no, phone number, Roll NO.

2) Composite :
It can be divided into subparts.
like: Name of person (middle name, first name)

3) Single - valued.
only one value attributes
ex:- student ID, loan number.

4) Multi - valued.
Attribute having more than one value.
ex: phone no., nominee name.

5) Derived.
value of this type of attribute can be derived from the value of other related attributes.

ex: Age (age can be derived from DOB by calculating)

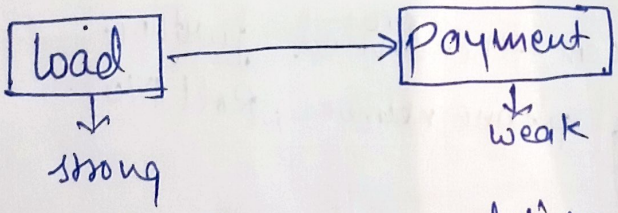
6) NULL value:

An attributes take a null value when an entity does not have a value for it.
mean "not applicable" or value not exist.
ex:- middle name.
or maybe the value is required but not given.

Relationships :-

→ weak entity & strong entity.

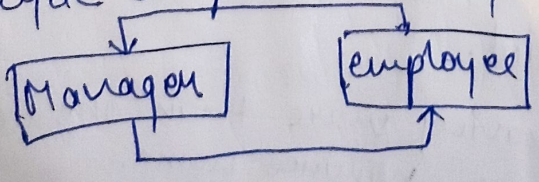
- strong entity can be identify by a unique id. it does not depend on any other entity.
- weak entity does not have any unique id. it is dependent on strong entity. for existence.



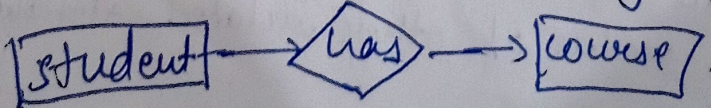
Relationship :- describe the relation among one or two or more than two entities

→ Degree of Relationship:

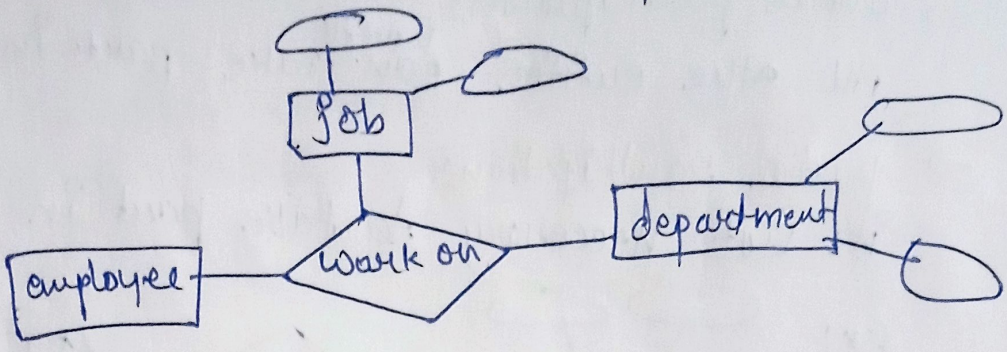
1) unary relationship.
only of cycle is given among one entity.



2) Binary relationship:-
two entities are involve in Binary relationship



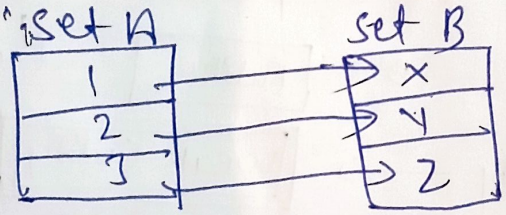
3) Ternary relationship.
It involve three entities among in a relation.



⇒ Relationship constraints.

1) Mapping cardinalities.
it define the no. of entities from set A can related to set B.

→ one to one.
only one entity can relate to one entities of set B
ex:- citizen adhaar. (each citizen hold only one adhaar card)



→ Many to one.
many entities of setA can related to one entities of set B.
ex:- courses taken by professar.

→ one to many.
one entity of setA can related to many entites of set B.
ex:- owner brought car.

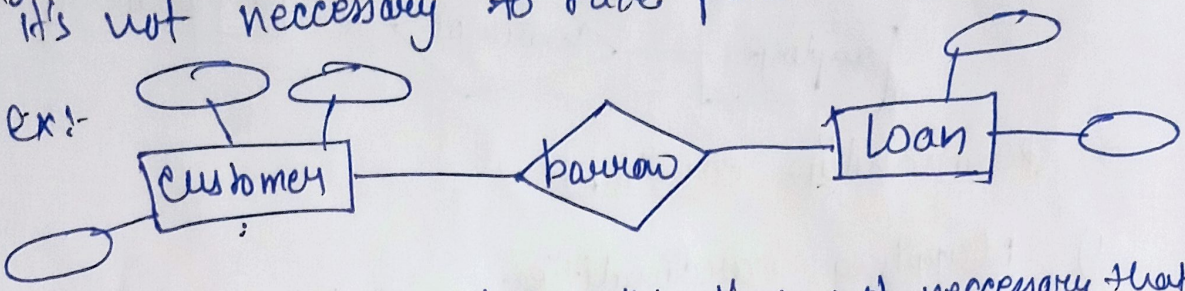
↔ Many to Many.
many entities of setA can relate to many entities of set B.

2) participation constant.

→ total participation:-
All the entities have to take part in relationship

→ partial participation:-
it's not neccessary to take part in relationship

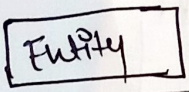
ex:-

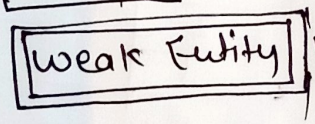


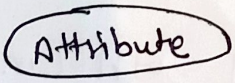
Customer → loan (partial participation:- not neccessary that customer borrow a loan)

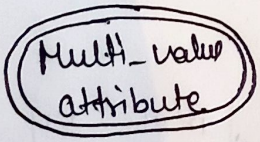
loan → customer (total participation; if load is there means ustomer buy the loan).

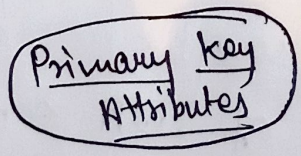
Symbols for ER Diagram.

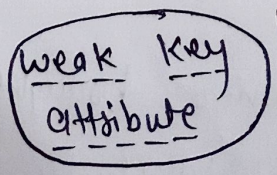
1) Entity → 

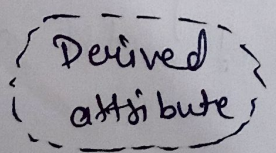
2) weak entity → 

3) Attribute → 

4) Multi-value attribute → 

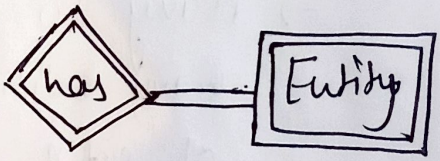
5) Primary key → 

6) weak key attribute → 

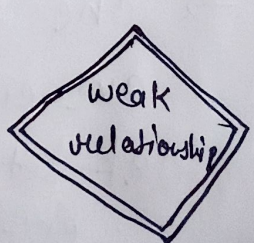
7) Derived attribute → 

8) Relationship → 

9) total participation



10) weak Relationship



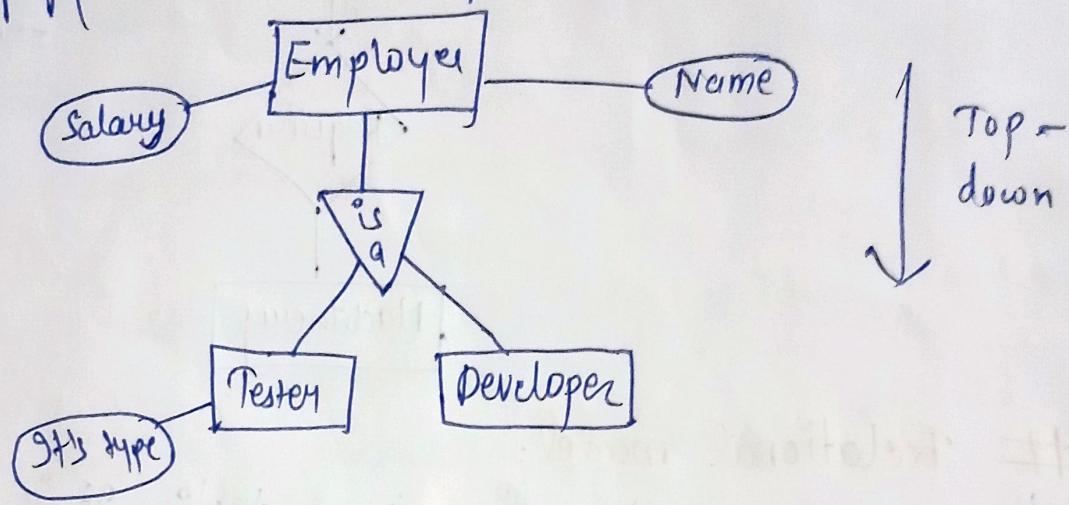
Specialization

An entity is divided into sub-entities based on its characteristics.

It is a top-down approach.

It simplify the visual representation of ER Diagram

ex:-

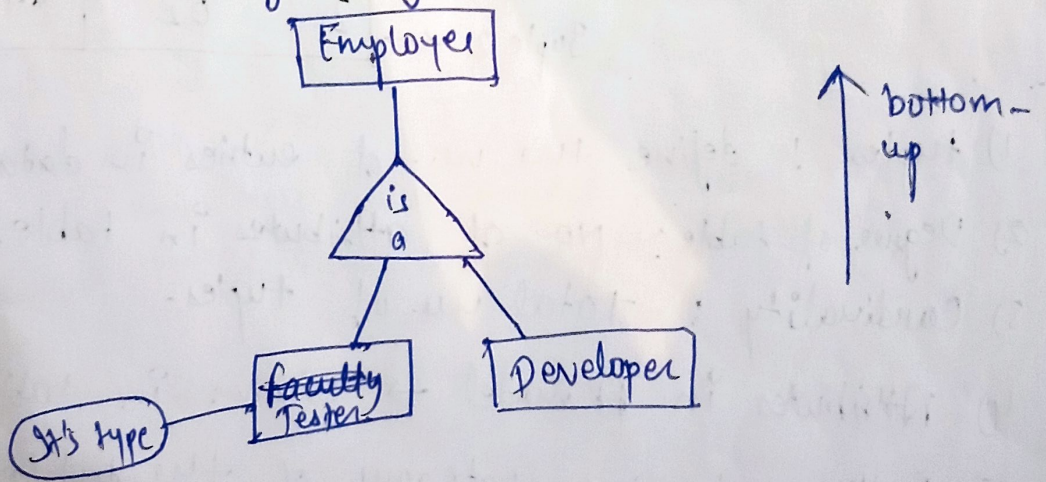


Generalization.

It is same as specialization, but here the approach of creation is from bottom-up approach.

It ~~eg~~ extracting common properties from a set of entities and creating a generalized entity from it.

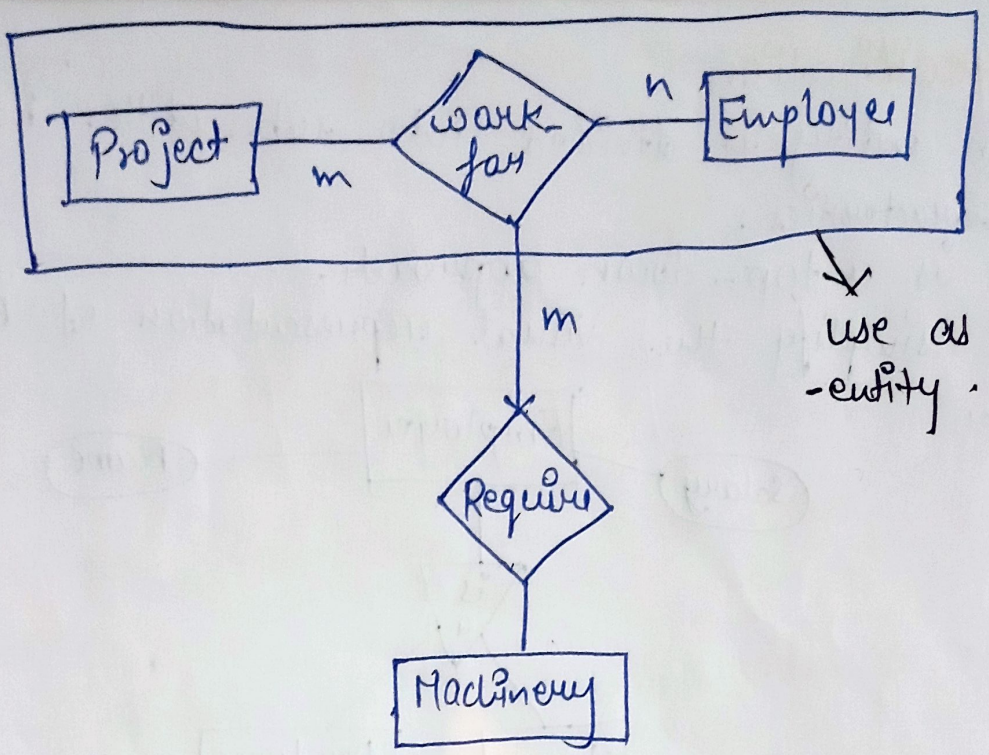
ex:-



Aggregation

An ER diagram is not capable of representing the relationship between an entity. In those case, a relationship with its corresponding entities is aggregated into a higher-level entity.

ex :-



Relational model :-

- Relations are two dimensional tables, It is easy to implement and easy to simplification in the operations to manipulate the data.

It uses primary key, secondary key.

Table → Relation →

	Cust_ID.	Name	add
Tuple 1 ← 1	01	ABC	PQ
Tuple 2 ← 2	02	XYZ	WZ

1) tuples : define the no. of entries in data.
2) Degree of table : No of attributes in table.
3) Cardinality : total no. of tuples.
4) Attributes in ER model → columns in table.
5) Entities is the category of the table
6) Relational key : set of attributes which can uniquely identify an each tuple.

Relational model key.

1) Primary key:-
it is a unique key. It can identify only one tuple at a time. It has no. duplicate values, it has unique value, It cannot be NULL. At last more than one column can also be a primary key for a table. ex: student_phone no, student_Id.

2) Super key:-
A set of attributes that can uniquely identify a tuple is known as super key.
ex: student_Id, (student_Id, student_name), (studen_Id, student_contact)

3) Candidate key:-
The minimum set of attsibutes that can uniquely identify a tuple is known as a candidate key.
→ In super key (student_Id, stud_name, stud_contact) can be
→ a set, but in candidate key (student_Id, stud_contact) is a set of attsibute cuz name can be same of more than one person.
→ And in primary key only student_Id can be a primary key there it identify every tuple unique in minimum attsibutes.
→ Candidate key shouldn't be NULL

4) Alternate key:-
All the leftout attsibutes in ~~cad~~ candidate key after selecting the primary key is alternate key.
ex:- (student_contact) is alternate key.

5) Foreign key :-

It creates relation between two tables.
A relation, say r_1, may include among its attributes the PK of an other relation, say r_2. This attribute is called FK. Foreign key from r_1 referencing r_2.
where we add foreign key that table is known as child table / Referencing relation. and r_2 is called Referenced (Parent) relation of foreign key.

Table 1

order_ID	Product	price	unique_id
1 1 1	—	—	1
1 2 2	—	—	2
1 3 3	—	—	1

child table

Table 2

unique_id	Name	contact	email
1	—	—	—
2	—	—	—
3	—	—	—

Parent table.

6) Composite key :-

To uniquely identify rows of table, a combination of two or more attributes can be used.
ex :- (fullname + DOB)

7) Compound key :-

A primary key which formed using 2 Foreign key

8) Surrogate key :

when we merge two different type of Database where the unique_id is defined different way in both then Database automatically generate a surrogate key, usually a integer value which is a primary key for identify tuples in merged table.
surrogate key may be used as primary key.

Integrity constraints :-

Integrity constraints are a set of rules. It is used to maintain the quality of information.
Integrity constraints ensure that the data insertion, updating and other processes have to be performed in such a way that data integrity is not affected.
Thus, Integrity constraints is used to guard against accidental damage to the database.

1) Domain constraints :-
defined as the definition of a valid set of values for an attribute, example id → int, name → char, phone → num.

2) Entity Constraints :
This entity integrity constraint state that primary key value can't be null.

3) Referential Constraints
→ In the Referential Integrity constraints, if a foreign key in Table1 refers to the primary key of table2, then every value of foreign key in Table1 must be null or be available in Table2.
→ value can't be inserted in child table if the value is not lying in parent table.
→ we can't delete the value of parent class if value is lying in child table.
* we can delete value from parent table if the value is lying in the child table ~~auto~~ ~~no~~ without violeting ~~not~~ delete constant. (On delete cascade).

```
Create table order ( order_ID int, Primary key, ..., cust_ID int
                     referencing customer on delete cascade)
```

This state that, whenever a tuple delete from customer it's also delete the foreign ID's value from order table

ON Delete NULL :-
In this we do not delete the values in order table while deleting customer table. we just initialize the corresponding foreign key as NULL.

4) Key constraints:-

(i) Not null :- it define that the value of the attribute cannot be null. ex :- primary key's values.

(ii) unique :-
ensure all value in column are different. Both unique & primary key constraints provide unique values. but you may have many unique constraint per table (contact, email) but only one primary key in a table.

(iii) Default constrain :-
set default value of column as "0", it can be only 0 & 1.

(iv) check constrain :-
we define a limit in a Domain/column.
ex :- check (age >= 18)

(v) primary key :-
same as ~~prior~~ privious discussed.

(vi) Foreign key constraint :-
keep relation b/w 2 table.

MySQL / SQL

RDBMS (Relational Database managment system).

An RDBMS is a type of database system that organizes data into tables with rows and columns. It uses a structured query language (SQL) for data manipulation, enforces data integrity through constraints, supports ACID properties for transaction reliability, and enables relationship between tables.

some of the RDBMS are:-

- MySQL
- SQL server
- oracle

Structured Query Language (SQL)

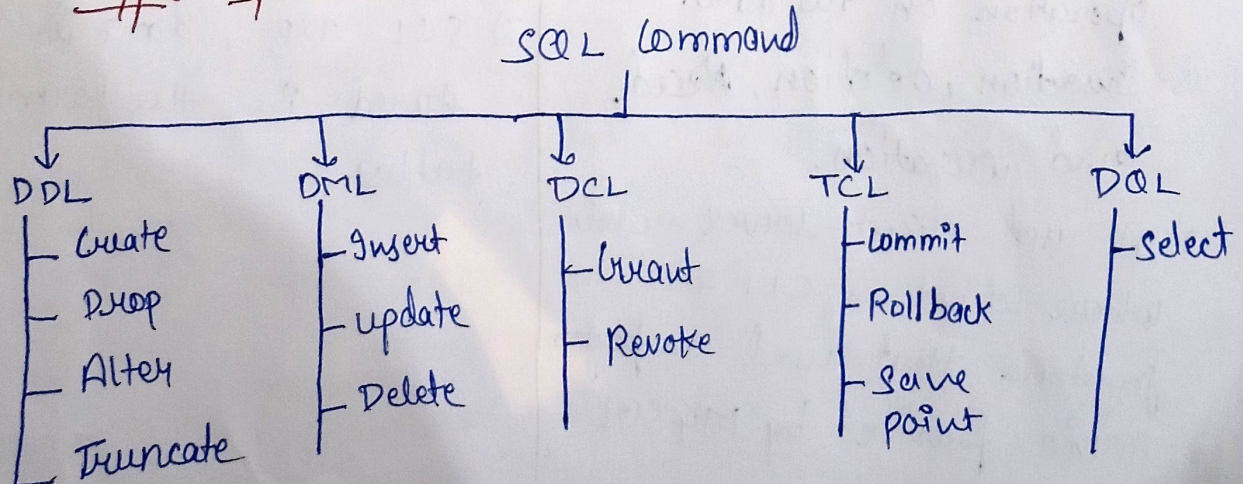
It is a domain specific language used for operations like querying, inserting, updating, and deleting data in a structured and organized manner. SQL is essential for interacting with RDBMS.

MySQL	SQL
1) It is a RDBMS.	1) It is a query language.
2) It use SQL to perform operation on the data. Insertion, deletion, Read, and creation.	2) It is a way to access data.
3) It used client-server model, where client is CLI or frontend that used ~~to perform~~ service provided by MySQL server	3) SQL DB, data is stored in the form of tables.

SQL DATATYPES

1) Char (string [0-255], example char(255))
2) Varchar (string (0-255) [better than char coz it use less memory])
3) Tinytext (string [0-255])
4) Text (string [0-65535])
5) BLOb (string [0-65535]) used to save files like audio, vedio.
6) Mediumtext (string [0-16777215])
7) MediumBlob (" ")
8) Longtext (string [0-4294967295])
9) Longblob
10) Tinyint (integer [-128 to 127])
11) smallint
12) Mediumint
13) INT, 14) BIGINT, 15) float, 16) Double, 17) Date, 18) Datetime, 19) Time, 20) ENUM, 21) set
22) Boolean 23) Bit

Type of SQL command



1) Data Definition language (DDL)
Structure of the table like, create, delete, altering a table. It save permanent.

2) Data manipulation language (DML)
used to modify the database. It is not permanent change in DB

3) Data control language (DCL)
used to grant and take back authority from any database user.

ex:- ① GRANT SELECT, UPDATE ON My_table TO some_user;

② Revoke SELECT, UPDATE ON My_table from USER1, USER2;

GRANT (grant permission to user)
REVOKE (take back permission)

4) Transaction control language (TCL)
To manage transactions done in the DB

→ START TRANSACTION: begin a transaction
→ COMMIT: apply all the changes and end transaction
→ ROLLBACK: Discard changes and end transaction.
→ SAVEPOINT: checkout within the group of transactions in which to rollback.

5) Data Query language (DQL / DRL)
DQL is used to fetch the data from the DB.

ex:- SELECT emp_name
FROM employee
WHERE age >20;

DRL operations :-

① Syntax : SELECT <set of column names> FROM <table-name>;

→ We can use SELECT keyword without using FROM by using DUAL table (which is a dummy table created by RDBMS).

ex:- SELECT 43+9 // return sum in grid form

SELECT now(); // give the DB current time.

SELECT ucase(); // provide uppercase letters.

② WHERE

Reduce rows based on conditions.

Ex: SELECT * FROM employee where SALARY >10000;

③ BETWEEN

SELECT * FROM employee WHERE age between 0 AND 30;

[0 and 30] both are also consider.

④ IN

It reduces OR conditions;

ex:- SELECT * FROM employee WHERE name IN ('ROSH', 'Roshan');

⑤ AND/OR/NOT

AND → WHERE condition_1 AND condition_2;

OR → WHERE condition_1 OR condition_2;

NOT → WHERE ~~condition~~ employee_id NOT IN (1,2,3,4);

⑥ SORTING entries.

ex:- ① SELECT * FROM employee ORDER BY SALARY;

② SELECT * FROM employee ORDER BY SALARY DESC;

[Default : ASC]

⑦ DISTINCT. Values in data/column.

It return the no. of different department/ categories in a columns.

ex:- no. of Department in company.

IT
HR
SALES. etc.

→ SELET DISTINCT department from employee;

⑧ Data Grouping

It group the categories by using some conditions.

SELECT department, FROM employee GROUP BY department;

COUNT(*)

output →

department	count(*)
HR	2
SALES	1
IT	3

→ Collect data from multiple records and group the result by one or more column.

→ used with aggregation function to perform various actions.

1) count()
2) SUM()
3) AVG()
4) MIN()
5) MAX()

9) GROUP HAVING

→ out of the categories made by GROUP BY, know only particular thing (cond).

→ similar to where

→ SELECT DEPARTMENT, COUNT(department) FROM employee group by Department having count(department) >2;

WHERE	HAVING
1) use to filter the rows from the table based on specified condition.	1) used to filter the rows from the groups based on specified condition.
2) where is used before GROUP BY clause	2) HAVING is used after GROUP BY.
3) WHERE can be used with SELECT, UPDATE & DELETE	3) GROUP BY is necessary for using HAVING.

DDL (Data Definition Language)

→ constraints.

1) Primary key:-
It can't be NULL, it is unique and only one per table.

2) Foreign Key:-

→ It refers to the Primary key of other table.
→ each relation can having any number of foreign key.

Syntax: FOREIGN KEY (cust_id) REFERENCES Customer (id)

3) UNIQUE

use to define a ~~person~~ row as a unique values in it.

Syntax: Name varchar(25) UNIQUE,

4) CHECK

use to apply an condition for a particular row., if it not satisfy the condition entry is denied.

Syntax:- balance INT,
CONSTRAINT acc_balance_chk CHECK(balance>1000)

→ balance should be greater than 1000.

5) DEFAULT

set default value of the column.

Syntax:-

amount DOUBLE NOT NULL DEFAULT=0,

6) ALTER OPERATIONS

a) Add new column

Syntax: ALTER TABLE Customer ADD age INT NOT NULL;

b) MODIFY
change datatype of an attribute.
Syntax: ALTER TABLE Customer MODIFY name char(255),

c) CHANGE COLUMN
Rename column name.
Syntax: ALTER TABLE customer CHANGE COLUMN name Customer-name VARCHAR(255);

d) DROP COLUMN
Drop a column completely.
Syntax: ALTER TABLE customer DROP COLUMN middle-name;

e) RENAME
Rename table name itself.
syntax: ALTER TABLE customer RENAME TO customer-details;

Data manipulation language (DML)

1) INSERT
Insert into table-name (col1, col2, col3) values (v1, v2, v3);

2) UPDATE
Update the presented data in table.
syntax: UPDATE student SET name = "Rosheni" WHERE id = 1;

3) ON UPDATE CASCADE
If we have two table and the primary key of one table is the foreign key of second table, if we make any change in primary column then by using ON UPDATE CASCADE the change will be also happens second table as well.

4) DELETE

DELETE FROM table_name WHERE id = 2;

DELETE FROM table_name; (all rows will be deleted).

5) DELETE CASCADE

if parent table entry is deleted than what about children table entry.

Syntax:
```
CREATE TABLE ORDER(
order_id INT PRIMARY KEY,
delivery_date DATE,
cust_id INT,
FORBIGN KEY (cust_id) REFERENCES customer (id)
 ON DELETE CASCADE
);
```

6) REPLACE

used for already present tuple in a table.

→ used as update, using REPLACE with the help of WHERE clause in primary key, then that row will be replaced

→ used as insert, if there is no duplicate data new tuple will be inserted.

(as insert) ↓

Syntax :→ REPLACE INTO student (id, name) VALUES (5, 'ROSH');

(as update) → REPLACE INTO table SET col1 = val1, col2 = val2;

7) JOINING TABLE :-

→ We refer to other tables to get meaningful outcomes.

→ Foreign key are used to do references to other table.

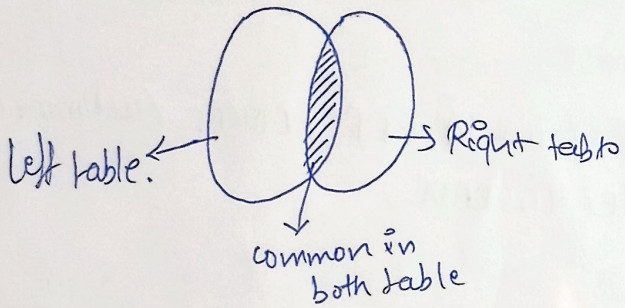
→ INNER JOIN

1) return a resultant table, which has matching values from both or all the tables.

2) · Syntax:

SELECT 'column_list FROM table_name1 · INNER JOIN table_name2 ON condition1 INNER JOIN table3 ON condition 2

3) for applying JOIN · operation, it's necessary to have a common attribute & between both the table.

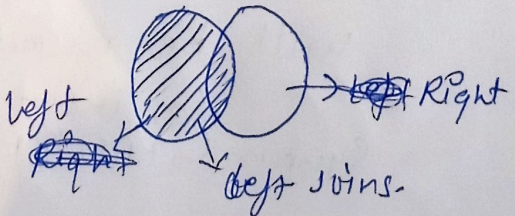


→ OUTER JOIN.

1) LEFT JOIN

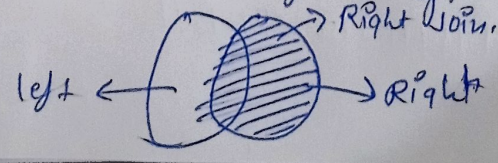
This returns a resulting table that all the data from left table and the matched data from the right table.

syntax: SELECT column FROM table LEFT JOIN table2 ON join_condition



2) Right JOIN.

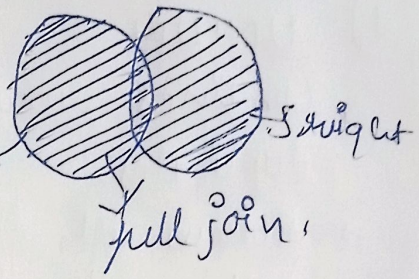
This return a resulting table that all the data from Right table and matched data from left table



Syntax: SELECT columns FROM table RIGHT JOIN table2 ON join_conditions.

→ FULL JOINS

" return a table that all the data from left and right table contain.



* There is no such key as full join, left

so, we evaluate, ~~left join~~

LEFT JOIN ∪ RIGHT JOIN

↓

union.

Syntax:

Select * from left_table as l LEFT JOIN Right_table as r

ON l.key = R.key

UNION

select * from left_table as l RIGHT JOIN right_table as r

ON l.key = R.key.);

→ CROSS JOINS:

It provide all the possible resultance of both the table.

Table L — 5 rows

Table R — 10 rows

⇒ Resultout :- 5 × 10 = 50 rows.

~~SELECT~~.

⇒ use join without using join keyword.

syntax:

Select * FROM lefttable, Rightable WHERE lefttable.id =

Right table.id;

SET operations.

based on concept of from set theory. allowing you to create new result sets with unique characteristic.

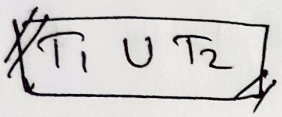
1) UNION

→ Combines rows from two or more queries, removing duplicates.

→ Return all distinct rows that appear in either or both queries.

syntax:-

SELECT * FROM table1 UNION ~~table2~~ SELECT * FROM table2.1



JOIN	SET operation
1) combine multiple tables based on matching condition.	1) combination is resulting set from two or more SELECT statement.
2) column wise combination.	2) ROW wise combination.
3) Data types of two table can be different	3) Data types of corresponding column from each table should be the same.
4) Can generate both distinct or duplicate rows.	4) generate distinct rows.
5) number of column(s) selected may or may not be the same from each table.	5) The number of column(s) selected must be the same from each table.
6) combines result horizontally.	6) combines results vertically.

2) Intersect :-

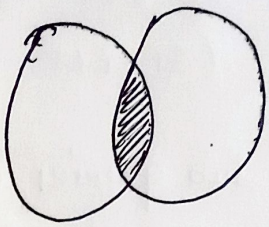
→ Return rows that are common to both queries.

→ Finds the intersection of two result sets.

syntax: SELECT * FROM tab1 INTERSECT SELECT * FROM Tab2;

→ This "intersect" word is not in SQL so, we emulate it with this syntax:

Select DISTINCT id from Tab1 INNER JOIN Tab2 using (id);



3) MINUS (or EXCEPT):

→ Return rows from the first query that are not present in the second query.

→ Finds the difference between two result sets.

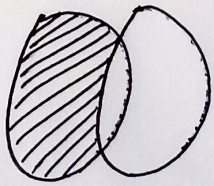
Syntax:-

SELECT * FROM tab1 · MINUS SELECT * FROM tab2.

→ again this is not in SOL, so again we emulate it.

→ SELECT id FROM Tab1 LEFT JOIN Tab2 using(id)

WHERE Tab2.id is NULL;



4) UNION ALL:

→ ~~Return rows~~

→ combines rows from two or more queries, including duplicates.

→ Return all rows from both queries, regardless of repetition.

Syntax:

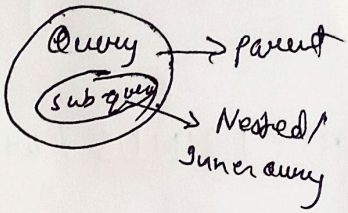
SELECT * FROM tab1 UNION ALL SELECT * FROM tab2;

⇒ SUB QUERIES:-

1) A subquery is a query nested within another query.
2) outer query depends on inner query.
3) Nested queries.
4) Syntax:-

```
SELECT col1 FROM table1 WHERE col_name OPERATOR
(SELECT col1 FROM table_name [WHERE]);
```

5) Sub queries exist mainly in 3 clauses.
   - → WHERE clause
   - → FROM clause
   - → SELECT clause.



6) using FROM clause

```
SELECT MAX(rating) FROM (SELECT * FROM movies WHERE
country = "India") as temp;
```

7) using SELECT clause.

```
SELECT (SELECT column_list FROM T_name WHERE condition),
column_list FROM T2_name WHERE condition;
```

Normalization:-

Firstly understand the Function Dependency (FD):-

⇒ Functional Dependency (FD):-

It states that the value of one attribute or set of attributes uniquely determines the value of another attribute or set of attributes.

* Denoted as $X \rightarrow Y$, where X is the determinant and Y is the dependent.

Ex:-

employee_ID	name	Dept.

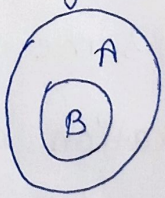
∵ employee_ID → name / employee_ID → Dept.

⇒ Types are:-

1) Trivial FD

A → B has trivial functional Dependency if B is a subset of A.

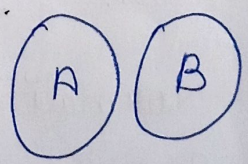
Ex:- A → A, B → B



2) Non-Trivial FD

A → B has a non-Trivial FD if B is not subset of A.

Ex:- A → B, A → C



⇒ Rules of FD (Armstrong's Axioms)

1) Reflexive:-

if A is subset of attributes and B is a subset of A.

Then A also find B within.

Ex:- A ⇒ {1, 2, 3, 4, 5} ⇒ A Reflexive B.

B ⇒ {1, 2, 3}

2) Augmentation.

If B can be determined from A, then adding an att. to this FD won't change anything.

[ex:- if $A \rightarrow B$, then $AZ \rightarrow BZ$.

3) Transitivity:

if $A \rightarrow B$, & $B \rightarrow C$, then $A \rightarrow C$.

⇒ Anomalies are:-

Anomalies means abnormalities, arises due to redundancy.

1) Insertion anomaly.

When certain data can not be inserted into the DB without the presence of other data.

ex:- Relation (stud_ID, stud_name, roll_no, Branch_ID, Branch_name)

can not add Branch Info, if student hasn't select any Branch.

2) Deletion anomaly:-

where the deletion of data results in unintended loss of some other important data.

3) updation anomaly (modification anomaly)

an update of a single data value require multiple rows of data to be update.

⇒ Normalization.

Due to, these anomalies, DB size is increase & DB performance became very slow. To rectify these anomalies we use Data optimisation technique called Normalization.

Normalization, the process of organizing data in a way that reduces redundacy and improves data integrity. involves breaking down tables into smaller, more focused table based on FDs.

⇒ Types of Normal forms

1) 1NF

Every relation cell must have atomic value, relation must not have multi-valued attributes.

2) 2NF

Relation must be in 1NF

There should not be any partial dependency.

- all non-prime attributes must be fully dependent on PK.
- Non-prime attribute can not depend on the part of the PK (primary key).

Relation(A, B, C, D)

{ A, B → primary key
  C, D → non-prime

✗ Partial dependency ⇒ B → C {B is a part of PK}

3) 3NF

Relation must be in 2NF

No transitivity dependency exists.

- Non-prime should not find a non-prime attrib.

{A, B, C} // A, B (PK) // C Non-prime.

A, B → C, C → B This is inappropriate.

4) BCNF (Boyce-Codd normal form)

Relation must be in 3NF.

FD: A → B // A must be a superkey

- we must not derive attribute from any prime or non-prime attribute.

Transaction in DBMS:-

A transaction is a set of logically related operation that accen and potentially modify data.

It's like a self-contained unit of work that ensure data consistency and accuracy, even if something goes wrong in middle.

example :-

transferring money from one account to another.

→ this are the steps are done in middle:-

1) Read : verify the balance in both accounts.
2) withdraw : Deduct the amount from the senders account
3) Deposit : Add the amount to the receivers account.
4) commit : Mark the transaction as complete, making the changes permanent.

ACID Property of transaction:-

To ensure integrity of the data, we require that the DB system maintain the following properties of the transaction.

1) Atomicity:-

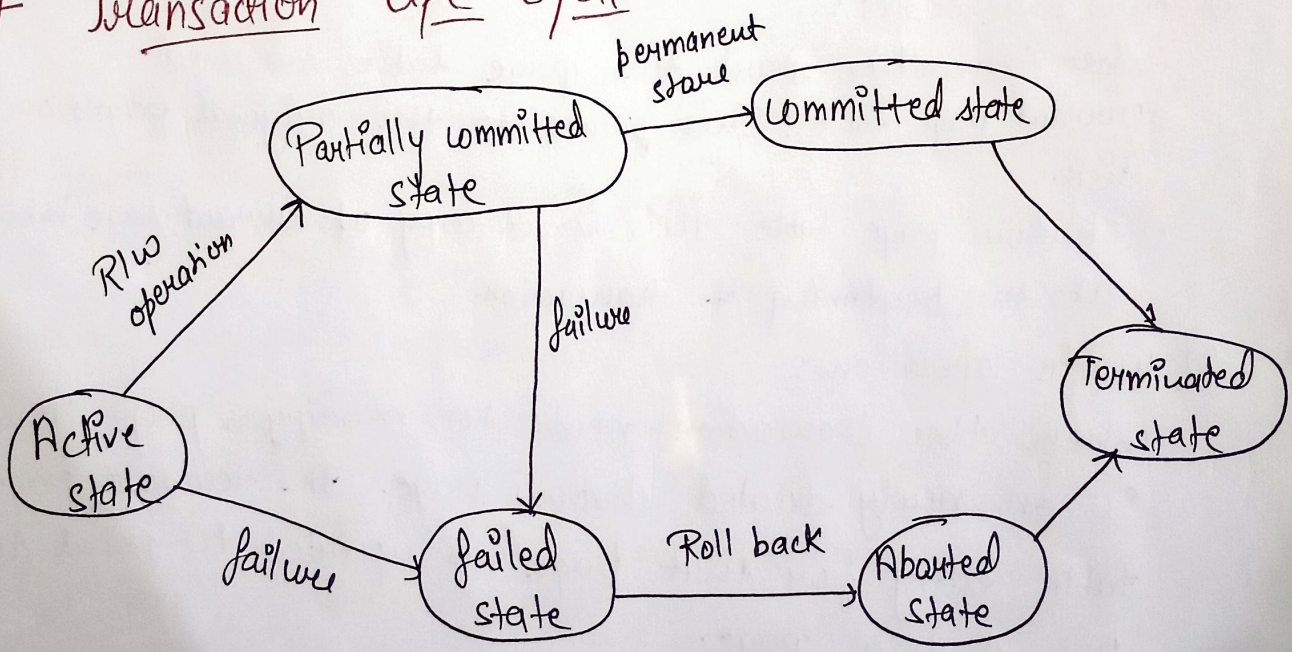
the entire transaction succeeds or fail completely. There's no partial completion.

2) Consistency:
The transaction ~~succeeds~~ brings the database from one valid state to another.

3) Isolation;
Concurrent transaction don't interfere with each other. Each sees the DB as if it's the only one running.

4) Durability:
Once committed, changes made by a successful transaction are permanent, even in case of system failure.

Transaction life cycle



1) Active state (read, write operation are being executed/performed)

2) partially committed state: After transaction executed the changes are saved in the buffer in the main memory. if changes are permanent it goes to committed state otherwise to failed state.

3) committed state :-

Shadow-copy scheme

→ Technique used in database system to improve transaction performance and ensure data consistency in case of crashes.

→ It's a copy-on-write technique, meaning that instead of updating data directly on the original page, a new "shadow page" is created and updates are applied there.

→ This ensure that the original data remains untouched until the transaction is committed successfully.

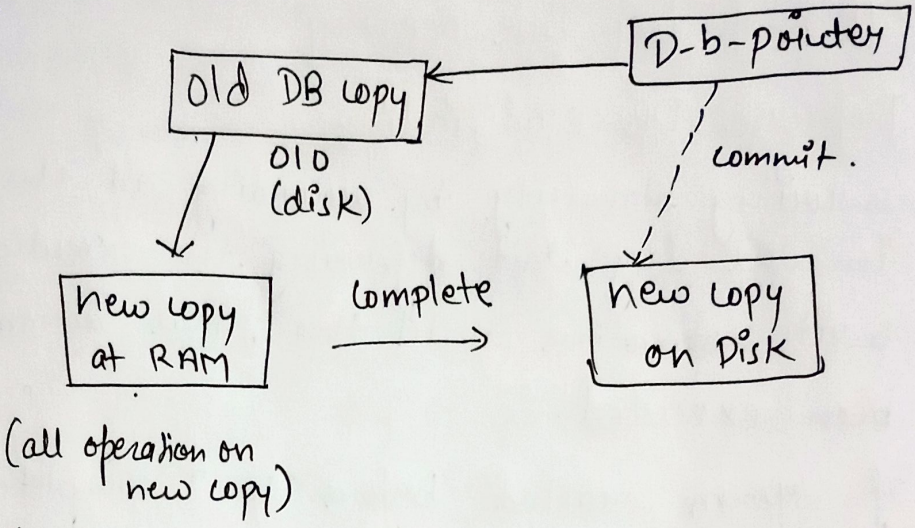
⇒ Work :-

1) Two page tables :-
when transaction start two page tables are used
→ current page table : This points to the original pages on disk.
→ shadow page table : This is a copy of current page table at the beginning of transaction.

2) Write operations :-
· when the transaction needs to modify a page, it does so on newly created shadow page. The current page table entry for that page is updated to point to the shadow page.

3) Transaction commit :-
If transaction commit successfully, the shadow page are copied back to the original pages, and the current page table is updated to reflect the changes.

4) Transaction rollback:
if transaction fail or needs to be rolled back, the shadow pages are simply discarded, and the current page table remain unchanged.



⇒ DRAWBACK :-
→ Increased storage space
→ Increased overhead.

Log. Based Recovery :-
It works by maintaining a chronological record of all all write operations performed on the database, called the transaction log.

1) Logging transaction operations:
records in the log file before being applied to the actual database.
It include details like transaction ID, operation type, data item affected, old and new values, etc.

2) Crash recovery :-
If system crashes during transaction, the log becomes crucial for recovery.
- Based on this analysis, the recovery perform two key actions:

a) Redo : for incomplete transaction with a "commit" record in the log,

b) Undo : for incomplete transaction without a "commit" record.

⇒ Types of log Base Recovery

1) Deferred DB Modification.

→ ensuring atomicity by recording all the DB modification in the log but deferring the execution of all the write operation until the final action of the T has been executed.

→ If system crashed before the T complete, or if transaction aborted, the info in the logs are ignored.

→ if transaction completes, the records associated to it in the log file are used in executed the deferred write

→ if failure occur while this updating is taking place, we perform redo.

2) immediate DB modification.

Updates the DB and writes to the log simultaneously. faster but riskier if the crash occurs before writing to the log.

In event of crash or Transaction failure, system uses old value field of the log records to restore modified values.

< write, 1000, 950 >
old new

INDEXING in DBMS

→ Indexing is used to optimise the performance of a Database by minimising the number of disk accesses required when a query is processed.

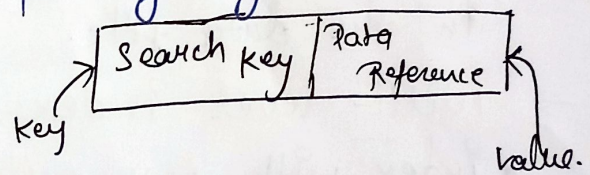
→ Type of DS, used to locate and access the data in a DB table quickly.

→ speeds up operation with read operations. like SELECT queries, WHERE clause etc.

→ Search key: contains copy of primary key or candidate key of the table or something else.

→ Data Reference: pointer holding the addrss of disk block where the value of the corresponding key is stored.

→ Index file is always sorted.



→ Indexing methods:-

1) primary Index (clustering Index):-

If the data file containing the records is sequentially ordered, a primary ~~key~~ index is an index whose search key also defines the sequential order of the file.

→ Dense And sparse indices:-

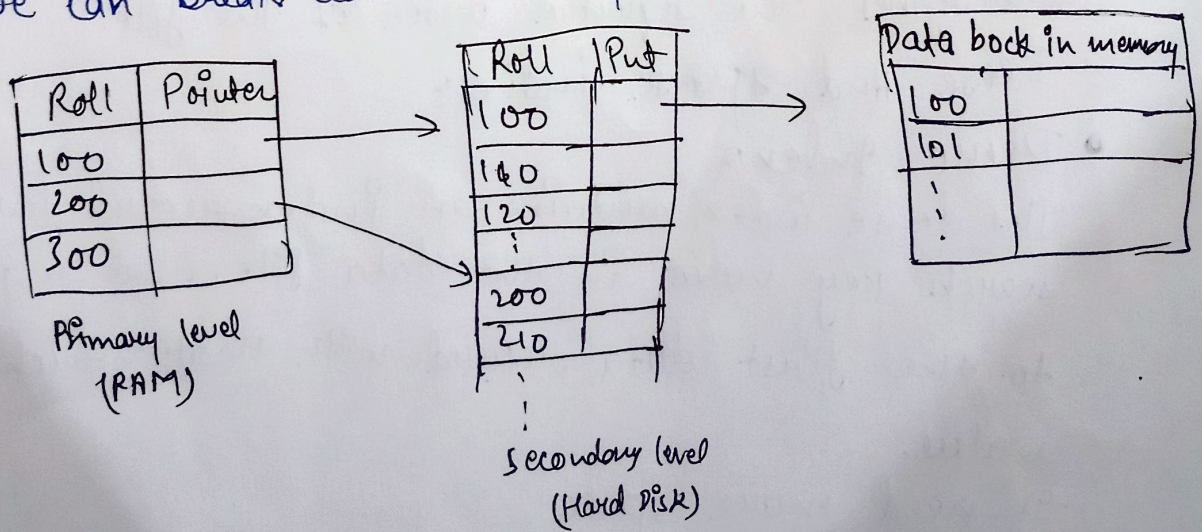
- Dense Index:-
  The dense index contains an index record for every search key value in the data file. and a pointer to the first data record with that search-key value.
  - It need more space
- Sparse Index
  An index record appears for only some of the search

Key values in the data file.
Solve issue of dense indexing in DBMS. In this method of indexing technique, a range of index columns stores the same data block address, and when data needs to be retrieved, the block address will be fetched.

→ Based on key ~~index~~ attributes :-
- primary key will be used as search-key in index.
- Sparse index will be formed, no. of entries in the index file = no. of blocks in datafile.

→ Based on Non-key attributes :-
- Datafile is sorted w.r.t non-key attributes.
- No. of entries in the index = unique non-key att. value in the data file.

→ Multi-level index :-
- index with two or more levels.
- If the single level index become enough large that the binary search it self would take much time, we can break down indexing into multiple levels.



INDEXING in DBMS

→ Indexing is used to optimise the performance of a Database by minimising the number of disk accesses required when a query is processed.

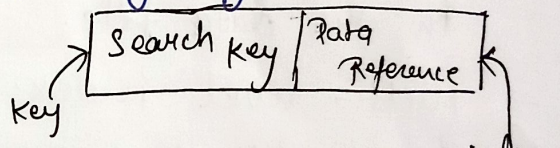
→ Type of DS, used to locate and access the data in a DB table quickly.

→ speeds up operation with read operations. like SELECT queries, WHERE clause etc.

→ Search key: contains copy of primary key or candidate key of the table or something else.

→ Data Reference: pointer holding the addrss of disk block where the value of the corresponding key is stored.

→ Index file is always sorted.



→ Indexing methods:-

1) primary Index (clustering Index):-

If the data file containing the records is sequentially ordered, a primary ~~key~~ index is an index whose search key also defines the sequential order of the file.

→ Dense And sparse indices:-

- Dense Index:-

The dense index contains an index record for every search key value in the data file. and a pointer to the first data record with that search key value.

- It need more space

- Sparse Index

An index record appears for only some of the search

2) Secondary Indexes:-

- Datafile is unsorted. Hence, primary indexing is not possible.
- can be done on key or non-key attribute.
- No. of entries in the index file = no. of records in data file.
- example of Dense index.
- It's benefits are that the index DS is in sorted, so we can apply binary search on that table instead of unsorted DS.

Type of DB

1) object-oriented datamodeling :-
Based on OOPs, Inheritance, object-identity, and encapsulation are used.
with methods to provide an interface to objects, are among the key concepts of object-oriented programming that have found application in data modelling.

→ Advantages :-
→ Data storage and retrieval is easy and quick.
→ can handle complex data relations and more variety of data types that standard relational databases.
→ Relatively friendly to model the ~~advantage~~ advance real world problem.
→ works with functionality of OOPs ~~and~~.

→ Disadvantage :-
→ High complexity causes performance issue like read, write, update and delete operations are slowed down.
→ not much of a community support as isn't widely adopted as relational databases.
→ Does not support views like relational databases.

2) NOSQL Database :-
→ It is non tabular database & store data differently.
→ It's types are Document, key-value, wide-column, and graph. They provide flexible schemas and scale easily with large amounts of data and high user load.
→ They are schema free.
→ They are more flexible & has the ability to adjust dynamically.

→ can handle huge amount of data.
→ Most off the NoSQL are open sources and has the capabity of horizontal scaling.

3) Hierarchical Database:-
→ most appropriate for use cases in which the main form of information gathering is based on a concrete hierarchy, ~~data can be retrieved~~.
→ The schema for hierarchical databases is defined by its tree-like organisation, in which there is typically a root "parent" directory of data stored as record that link to various other subdirectory branches.
→ The one-to-many structure is not ideal for complex structures as it cannot describe relationships in which each child node has multiple parents nodes.

4) Network Databases:-
→ Extension of Hierarchical databases.
→ The child records are given the freedom to associate with multiple parent records.
→ organised in a graph structure.
→ can handle complex relations.
→ Maintenance is tedious.
→ M:N links may cause slow retrieval.
→ not much web community support.
→ ex:- Integrated Data Store (IDS), IDMS (Integrated Database management system), TurboIMAGE etc:

Clustering / replica set :-

⇒ Replica sets :-

Create and maintains copies of the same data on multiple servers (replicas).

- Improved read performance :- Query can be directed to any replica, distributing the load.
- Increased availability: If one server fails, another replica can take over.
- Enhanced disaster recovery: Replicas in different location can protect against regional outages.

→ Drawback :-

- Increased storage cost: Every replica require additional storage space.
- Data consistency complexity: Mechanism are needed to ensure all replicas stay synchronized with update.
- write performance impact: updates need to be applied to all replicas, potentially slowing down writes.

⇒ Data clustering :-

combines multiple server (nodes) into a single logical unit to share resources and workloads.

- Improved scalability;
- Enhanced performance
- Increased fault tolerance

→ Drawback :-

- Increased complexity
- potential single point of failure

Content Delivery Network (CDN):-

A CDN is essentially a vast network of servers scattered across the globe. Imagine it like a highway system for your website's content.
content like, image, video get delivered from the server closest to them, not from your origin server (the one where website lives).

→ Few key tricks CDNs use to speed things up:-

1) caching: popular content gets stored on these edge servers, so users don't have to wait for it to travel from your original server every time.
2) Routing:- CDNs figure out the fastest path for content to reach each user based on their location and network condition
3) optimization: Image can be compressed, video can be transcoded to different formats; ~~etc~~

→ Benefits.

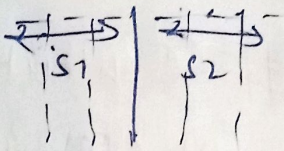
- faster loading times, improved SEO., reduce server load, Enhanced security.

Partioning and sharding in DBMS.

Partitioning is the technique used to divide stored database objects into separate servers. Due to this, there is an increase in performance, controllability of Data.
→ manage huge chunks of data optimally.

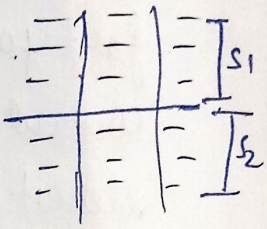
Types are:-

1) Vertical partitioning :-



→ slicing relation ~~horizontally~~ vertically / column wise.
→ Need to access different servers to get complete tuples

2) Horizontal partitioning :-



→ slicing relation horizontally / row wise
→ Independent chunks of data tuples are stored in different servers.

* we use partitioning when Dataset become much huge that managing and dealing with it become a tedious task.

or/

when number of requests are enough longer that the single Database server access Ps taking huge time and hence the system response time become high.

→ Advantages :-

1) parallelism
2) Availability
3) performance
4) Manageability
5) Reduce cost,

⇒ Distributed Database

A single logical database that is spread across multiple locations (servers) and logically interconnected by network.
This is the product of applying DB optimization technique like clustering, partitioning and sharding.

⇒ Sharding

Technique to implement Horizontal partioning.

→ The fundamental idea of sharding is the idea that instead of having all the data sit on one DB instance, we split it up and introduce a Routing layer so that we can forward the request to the right instances that actually contain the data.

→ It's cons:-

1) Complexity, making partition mapping, Routing layer to be implemented in the system, Non-uniformity that creates the necessity of Re-sharding.
2) Not well suited for analytical type of queries, as the data is spread across different DB instances.

CAP Theorem:-

CAP (Consistency, Availability, Partition Tolerance) Theorem is a fundamental principal in the world of distributed database system, including DBMS.

1) Consistency:-
all nodes see the same data simultaneously. The read operation ~~a~~ should cause all nodes to return the same data. All users see the same data at the same time, regardless the node they connect to.

2) Availability:-
It means that the system remains operational all of the time. Every request will get a response regardless of the individual state of the nodes. This means that the system will operate even if there are multiple nodes down.

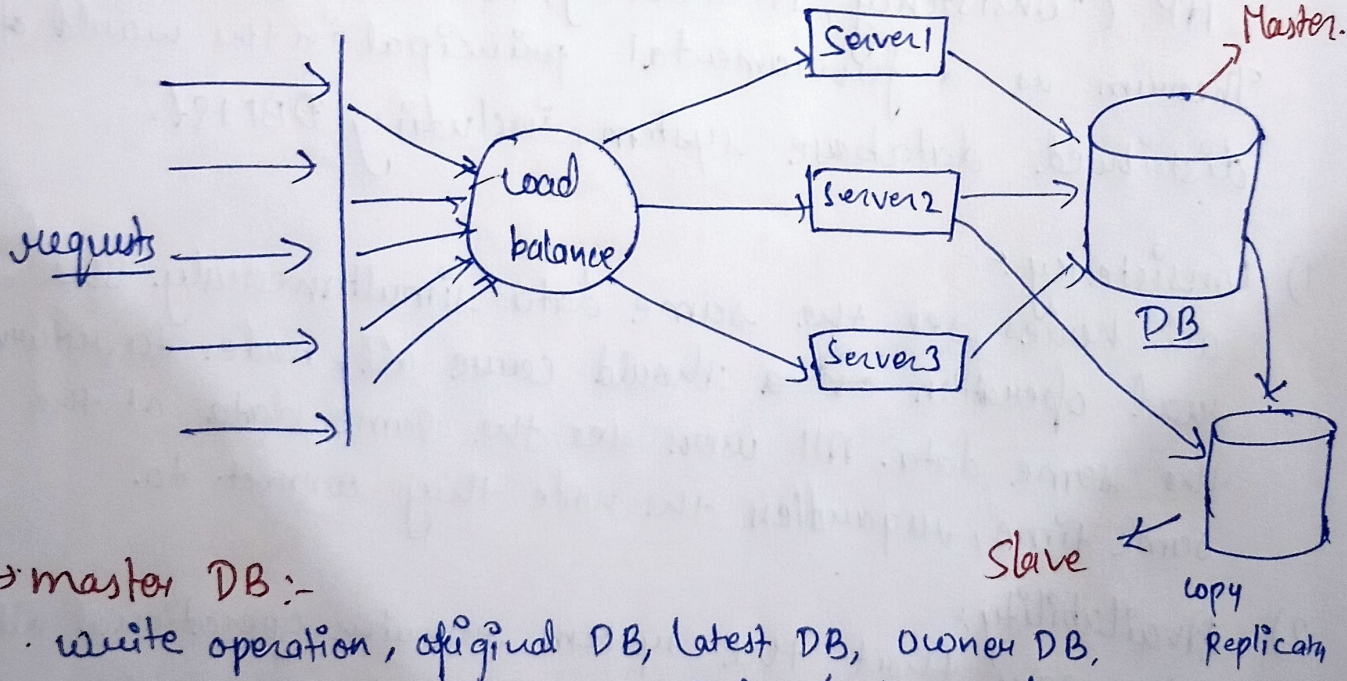
unlike a consistent system, there's no guarantee that the response will be the most recent write operation.

3) Partition Tolerance :-
It means that there's a break in communication between nodes. if a system is partition-tolerant, the system does not fail, regardless of whether msg are dropped or delayed between nodes within the system.

* The CAP theorem states that a distributed system can only provide two of three properties simultaneously

~~The~~ trade off between consistency and availability when there's a partition.

Master-slave Architecture :-



→ master DB :-
· write operation, original DB, latest DB, owner DB, primary DB, it has latest info. (latest update).

→ slave DB :
Read operation, DB replication.

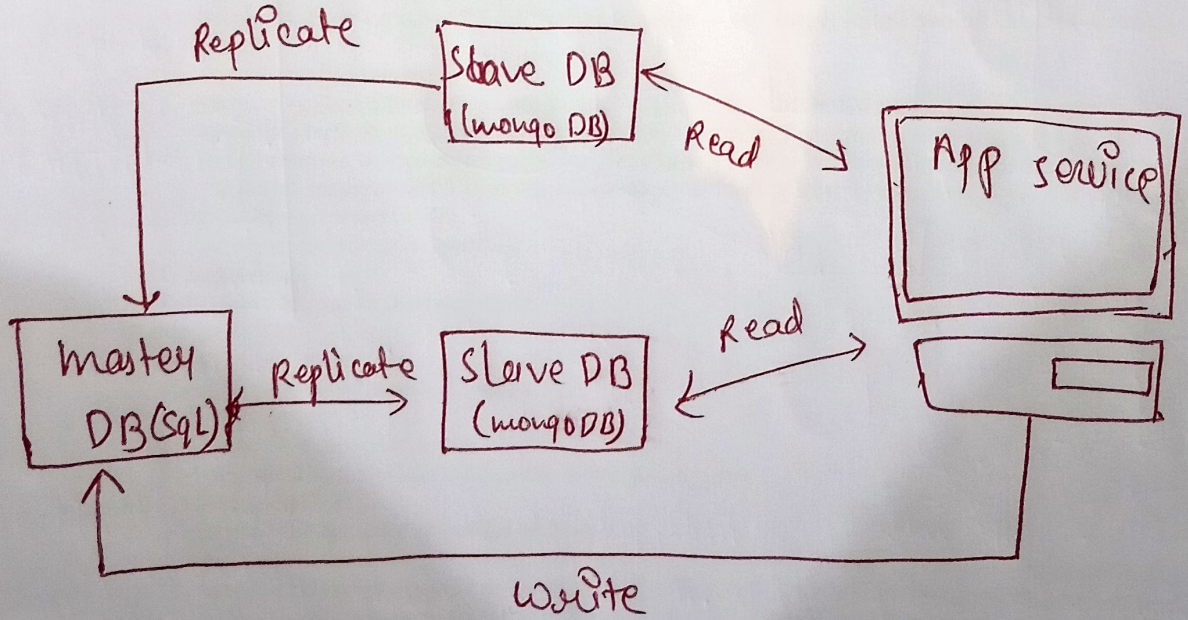
Qus if slave DB gets update query ?!

(1) ignore / never allow slave to accept request. slave only perform the read operation.

2) Allow it, & you have to make a way to propogate it to master. (but it's not master-slave model anymore).

⇒ Advantage :-

1) Back-up :- if master DB is down for sometime, atleast read operation is still happening.
2) Scale out Read operation.
3) Availability.
4) Reliability
5) latency reduce.



LEC-15: NoSQL

1. **NoSQL databases** (aka "not only SQL") are non-tabular databases and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph. They provide **flexible schemas** and **scale** easily with **large amounts of data** and **high user loads**.
    1. They are schema free.
    2. Data structures used are not tabular, they are more flexible, has the ability to adjust dynamically.
    3. Can handle huge amount of data **(big data).**
    4. Most of the NoSQL are open sources and has the capability of horizontal scaling.
    5. **It just stores data in some format other than relational.**
2. **History behind NoSQL**
    1. NoSQL databases emerged in the late 2000s as the cost of storage dramatically decreased. Gone were the days of needing to create a complex, difficult-to-manage data model in order to avoid data duplication. Developers (rather than storage) were becoming the primary cost of software development, so NoSQL databases optimised for developer productivity.
    2. Data becoming unstructured more, hence structuring (defining schema in advance) them had becoming costly.
    3. NoSQL databases allow developers to store huge amounts of unstructured data, giving them a lot of flexibility.
    4. Recognising the need to rapidly adapt to changing requirements in a software system. Developers needed the ability to iterate quickly and make changes throughout their software stack — all the way down to the database. NoSQL databases gave them this flexibility.
    5. Cloud computing also rose in popularity, and developers began using public clouds to host their applications and data. They wanted the ability to distribute data across multiple servers and regions to make their applications resilient, to scale out instead of scale up, and to intelligently geo-place their data. Some NoSQL databases like MongoDB provide these capabilities.
3. **NoSQL Databases Advantages**
    A. **Flexible Schema**
        1. RDBMS has pre-defined schema, which become an issue when we do not have all the data with us or we need to change the schema. It's a huge task to change schema on the go.

    B. **Horizontal Scaling**
        1. Horizontal scaling, also known as scale-out, refers to bringing on additional nodes to share the load. This is difficult with relational databases due to the difficulty in spreading out related data across nodes. With non-relational databases, this is made simpler since collections are self-contained and not coupled relationally. This allows them to be distributed across nodes more simply, as queries do not have to "join" them together across nodes.
        2. Scaling horizontally is achieved through **Sharding** OR **Replica-sets**.

    C. **High Availability**
        1. NoSQL databases are highly available due to its auto replication feature i.e. whenever any kind of failure happens data replicates itself to the preceding consistent state.
        2. If a server fails, we can access that data from another server as well, as in NoSQL database data is stored at multiple servers.

    D. **Easy insert and read operations.**
        1. Queries in NoSQL databases can be faster than SQL databases. Why? Data in SQL databases is typically normalised, so queries for a single object or entity require you to join data from multiple tables. As your tables grow in size, the joins can become expensive. However, data in NoSQL databases is typically stored in a way that is optimised for queries. The rule of thumb when you use MongoDB is data that is accessed together should be stored together. Queries typically do not require joins, so the queries are very fast.
        2. But difficult delete or update operations.

    E. **Caching** mechanism.

    F. **NoSQL** use case is more for **Cloud** applications.
4. **When to use NoSQL?**
    1. Fast-paced Agile development
    2. Storage of structured and semi-structured data
    3. Huge volumes of data
    4. Requirements for scale-out architecture
    5. Modern application paradigms like micro-services and real-time streaming.
5. **NoSQL DB Misconceptions**
    1. Relationship data is best suited for relational databases.
        1. A common misconception is that NoSQL databases or non-relational databases don't store relationship data well. NoSQL databases can store relationship data — they just store it differently than relational databases do. In fact, when compared with relational databases, many find modelling relationship data in NoSQL databases to be easier than in relational databases, because related data doesn't have to be split between tables. NoSQL data models allow related data to be nested within a single data structure.
    2. NoSQL databases don't support ACID transactions.

1. Another common misconception is that NoSQL databases don't support ACID transactions. Some NoSQL databases like MongoDB do, in fact, support ACID transactions.

6. **Types of NoSQL Data Models**
   1. **Key-Value Stores**
      1. The simplest type of NoSQL database is a key-value store. Every data element in the database is stored as a key value pair consisting of an attribute name (or "key") and a value. In a sense, a key-value store is like a relational database with only two columns: the key or attribute name (such as "state") and the value (such as "Alaska").
      2. **Use cases** include shopping carts, user preferences, and user profiles.
      3. e.g., Oracle NoSQL, Amazon DynamoDB, MongoDB also supports Key-Value store, Redis.
      4. A key-value database associates a value (which can be anything from a number or simple string to a complex object) with a key, which is used to keep track of the object. In its simplest form, a key-value store is like a dictionary/array/map object as it exists in most programming paradigms, but which is stored in a persistent way and managed by a Database Management System (DBMS).
      5. Key-value databases use compact, efficient index structures to be able to quickly and reliably locate a value by its key, making them ideal for systems that need to be able to find and retrieve data in constant time.
      6. There are several use-cases where choosing a key value store approach is an optimal solution:
         a) Real time random data access, e.g., user session attributes in an online application such as gaming or finance.
         b) Caching mechanism for frequently accessed data or configuration based on keys.
         c) Application is designed on simple key-based queries.
   2. **Column-Oriented / Columnar / C-Store / Wide-Column**
      1. The data is stored such that each row of a column will be next to other rows from that same column.
      2. While a relational database stores data in rows and reads data row by row, a column store is organised as a set of columns. This means that when you want to run analytics on a small number of columns, you can read those columns directly without consuming memory with the unwanted data. Columns are often of the same type and benefit from more efficient compression, making reads even faster. Columnar databases can quickly aggregate the value of a given column (adding up the total sales for the year, for example). **Use cases** include analytics.
      3. e.g., Cassandra, RedShift, Snowflake.
   3. **Document Based Stores**
      1. This DB store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values. The values can typically be a variety of types including things like strings, numbers, booleans, arrays, or objects.
      2. **Use cases** include e-commerce platforms, trading platforms, and mobile app development across industries.
      3. Supports ACID properties hence, suitable for Transactions.
      4. e.g., MongoDB, CouchDB.
   4. **Graph Based Stores**
      1. A graph database focuses on the relationship between data elements. Each element is stored as a node (such as a person in a social media graph). The connections between elements are called links or relationships. In a graph database, connections are first-class elements of the database, stored directly. In relational databases, links are implied, using data to express the relationships.
      2. A graph database is optimised to capture and search the connections between data elements, overcoming the overhead associated with JOINing multiple tables in SQL.
      3. Very few real-world business systems can survive solely on graph queries. As a result graph databases are usually run alongside other more traditional databases.
      4. **Use cases** include fraud detection, social networks, and knowledge graphs.

7. **NoSQL Databases Dis-advantages**
   1. **Data Redundancy**
      1. Since data models in NoSQL databases are typically optimised for queries and not for reducing data duplication, NoSQL databases can be larger than SQL databases. Storage is currently so cheap that most consider this a minor drawback, and some NoSQL databases also support compression to reduce the storage footprint.
   2. **Update & Delete operations are costly.**
   3. **All type of NoSQL Data model doesn't fulfil all of your application needs**
      1. **Depending on the NoSQL database type you select, you may not be able to achieve all of your use cases in a single database. For example, graph databases are excellent for analysing relationships in your data but may not provide what you need for everyday retrieval of the data such as range queries. When selecting a NoSQL database, consider what your use cases will be and if a general purpose database like MongoDB would be a better option.**
   4. **Doesn't support ACID properties in general.**
   5. **Doesn't support data entry with consistency constraints.**

8. SQL vs NoSQL

	SQL Databases	NoSQL Databases
Data Storage Model	Tables with fixed rows and columns	Document: JSON documents, Key-value: key-value pairs, Wide-column: tables with rows and dynamic columns, Graph: nodes and edges
Development History	Developed in the 1970s with a focus on reducing data duplication	Developed in the late 2000s with a focus on scaling and allowing for rapid application change driven by agile and DevOps practices.
Examples	Oracle, MySQL, Microsoft SQL Server, and PostgreSQL	Document: MongoDB and CouchDB, Key-value: Redis and DynamoDB, Wide-column: Cassandra and HBase, Graph: Neo4j and Amazon Neptune
Primary Purpose	General Purpose	Document: general purpose, Key-value: large amounts of data with simple lookup queries, Wide-column: large amounts of data with predictable query patterns, Graph: analyzing and traversing relationships between connected data
Schemas	Fixed	Flexible
Scaling	Vertical (Scale-up)	Horizontal (scale-out across commodity servers)
ACID Properties	Supported	Not Supported, except in DB like MongoDB etc.
JOINS	Typically Required	Typically not required
Data to object mapping	Required object-relational mapping	Many do not require ORMs. MongoDB documents map directly to data structures in most popular programming languages.