Hello everyone,

these are my notes for Database Managenent Systems, gave a lot of interviews and after multiple updates I can say that almost everything you will be asked in interviews is here.

Please let me know if you come across something that is incorrect or can be improved. You can reach me at

Linkedin: **https://www.linkedin.com/in/notabhishek/** Follow for more

Youtube: **https://www.youtube.com/channel/UCLPT3aCJwFPWsVjEPqnOYPQ**
Please subscribe so that you don't miss the content about to come

The end goal should be to able to revise within 15-20 minutes before an interview

PS: The only thing I can remember I was asked out of these were whose job is it to implement ACID? (DBMS/Database/Programmer), How would you implement isolation/ isolation levels.

Database Management System
(DBMS)

1. What is DBMS? How it is Different from Data Structures. 2. Difference Between Database and Database Management System? 3. Main Memory and Secondary Memory. 4. Basic Functionality and Advantages of DBMS i.e. Data Retrieve, Delete, Insert

(Functionality and Remove Redundancy, Easy access (Advantages) 5. DBMS Two-Level and Three-Level Architecture. 6. **Entity-Relationship Model** (ER Model, Know the Basics i.e What is Entity, Attributes, and Cardinality) (Hint: They won't ask You to make a Complex ER diagram) 7. **Relational Model** (Difference b/w ER model and Relational Model) 8. All types of Keys in Relational Model (Primary Key, Super Key, Candidate Key, Foreign

Key) (**Hint: Important**) 9. Armstrong Axioms and All Types of Functional Dependencies (Transitive and Relative) 10. Database Anomalies (Insert, Delete and Update) 11. **Normalization -> What is Normalization**? Why do we need Normalization ? 12. All the Types of Normalization, **1NF, 2NF, 3NF, and BCNF**. (Hint: They won't ask you to Convert the Functions into 1NF, 2NF, and All, Just Know the Basics of Each Normal Form & It Would Be Better if you Understand Every NF Form with an Example) (**Important topic**) 13. SQL Queries (Learn with one example of Each Query, Not Much, **Best Resource :**

SQL w3 Resource) 14. File Structures in Database (Indexing, Sparse Indexing, **B-Tree, B+ trees**) (Source

GFG) 15. Transactions, Operations of Transactions, **Transaction's ACID Properties**

(Important) 16. State of Transactions i.e Committed, partially Committed 17. **Concurrency Control**, Lock and all over Transactions

1. What is DBMS? How it is Different from Data Structures.

a. **DBMS** Database Management System
   i. **Database**
      1. Collection of interrelated data, allows the efficient insertion, retrieval, and deletion of data
   ii. **DBMS**
      1. The software used to manage databases is known as DBMS
      2. DBMS provides the following functionality
         a. Data definition
         b. Data retrieval
         c. Data updation
         d. User administration - monitor users, performance, enforce data security,
      3. E.g. MySQL, Oracle, etc.
   iii. DBMS vs File System - below are problems with File System, which are solved by DBMS Read more
      1. Redundancy of data
      2. Inconsistency of data
      3. Difficult data access
      4. Unauthorized access
      5. No concurrent access
      6. No backup and recovery
   iv. DBMS vs Data Structures Read more
      1.

DBMS	**Data Structures**
Organized collection of data	The special format of storing data to perform some operations efficiently
Non Volatile	Volatile
Types: Relational, NoSQL, centralized, hierarchical databases	Types: Linear and nonlinear data structures
Eg. MySQL, Oracle, MongoDB, etc.	Eg. Stack, Queue, Heap, Tree, Graph

---

2. Difference Between Database and Database Management System? [Read more]

a. **Database vs DBMS**
   i. A Database is a collection of interrelated data
   ii. DBMS is a software/set of software that help us manage the database
   iii.

	Database	**DBMS**
Storage	A Database can be maintained on paper/computers	DBMS maintains a database on computer
Data Retrieval	Can be done using prog languages like C++, Python, etc.	Uses query languages like SQL
Speed	Slow speed without query languages	Fast as it uses query languages
Access	Not a lot of people can use it at the same time	Provides functionality for multiple users at the same time(handles transactions)
Backup and Recovery	No backup/ recovery in case of failure	Maintains backup, and can recover in case of failure

---

3. Main Memory and Secondary Memory. [Read more]

a. **Memory**
   i. Computer memory is the storage space where data to be processed and instructions to run are stored
   ii. Two types
      1. **Primary memory**
         a. The main memory of the computer
         b. Smaller, Faster, Costly, Volatile
         c. Two types
            i. **RAM**

1. Random Access Memory
2. Volatile

ii. **ROM**

1. Read Only Memory
2. Non Volatile, Content is written by the manufacturer and cannot be changed

2. Secondary memory

a. Large, Slower, Inexpensive, Non-volatile

---

4. Basic Functionality and Advantages of DBMS i.e. Data Retrieve, Delete , Insert (Functionality) and Remove Redundancy, Easy access (Advantages) Read more

a. Features of DBMS

i. DBMS provides the following functionality

1. Data definition
2. Data insertion
3. Data retrieval
4. Data updation
5. Data deletion

b. Advantages of DBMS

i. Removes data redundancy
ii. Data sharing - levels of authorization for sharing
iii. Data Integrity - data is correct at all times inside the DB
iv. Data Security/ Privacy - only authorized users can access
v. Data Consistency -

1. Data is the same for all users at all times
2. Any change in DB is reflected to all users

c. DDL vs DML GfG

i. DDL: data definition language, ex: create, alter, rename, drop
ii. DML: data manipulation language, ex: insert, update, merge

---

5. DBMS Two-Level and Three-Level Architecture. Read more

a. DBMS Architecture

i. Two-tier

1. Basic client-server model
2. Easy to manage/implement
3. Users set up an individual connection with the server
4. Poor performance in case of a large number of users
5. Poor security since everyone accesses the actual server

ii. Three-tier

1. Client - Application Layer - Server
2. Harder to implement and manage
3. Users interact with the Application layer(middle layer)
4. The application layer acts as a medium for the exchange of partially processed data
5. Enhanced scalability by the distributed deployment of application layer
6. Better data security, since the user only interacts with the application layer and not with the server
7. Improved data integrity since data corruption can be avoided using a middle layer

---

6. **Entity-Relationship Model** (ER Model, Know the Basics i.e What is Entity, Attributes and Cardinality) (Hint: They won't ask You to make a Complex ER diagram) Read more

a. Entity-Relationship Model Video

i. It is a **conceptual/ logical model**
ii. *Logical representation* of entities and relationships

iii. Entity

1. An object which has a physical existence
2. E.g. Student, Course, etc

iv. Entity type

1. An Entity is an object of Entity type
2. E.g. E1 is an entity of Entity type Student

v. Entity set

1. Set of all entities of given Entity type
2. E.g. set of students

vi. Relationship

1. Association between two or more entities

vii. Attributes

1. Properties/ Characteristics that define an entity type
2. Just like column names

viii. Entity Type or Schema

1. Student(Roll no, Age, Address)

ix. Representation

1. Entity 

2. Relationship 

3. Attributes 

x. Types of attributes Video
   1. Single Valued registration number
   2. Multi-Valued Mobile Number 

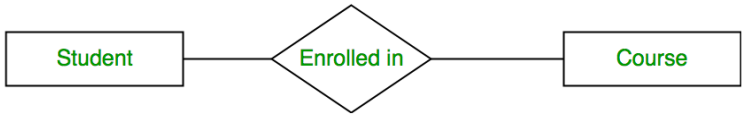
   3. Simple
   4. Composite Name(composite) -> fname, mname, lname
   5. Stored
   6. Derived DOB-> age (derived by using dob)

      a.
      b. Represented by dotted circle/oval
   7. **Key** -> unique, can uniquely identify row

      a.
      b. Underlined
   8. Non-Key
   9. Required vs Optional attribute
   10. Complex - composite + multi-valued
      a. eg. address(city, state, country) composite
      b. address (home address, office address) multi-valued

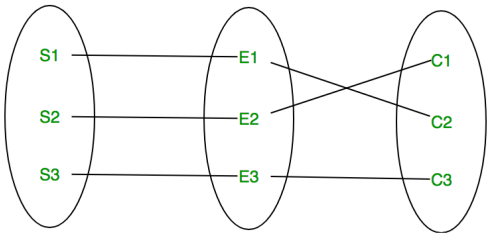
xi. Relationship type and Relationship set

1. A relationship represents an **association** between **entity types**

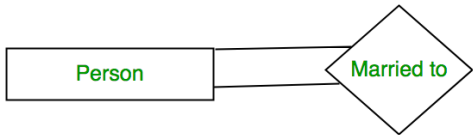


2.

3. Represented using diamond
4. A set of relationships of the same type is known as **Relationship set**

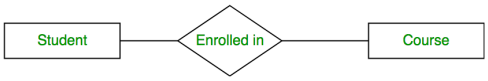


5. E.g.
6. **Degree of Relationship set**
    a. Number of different entity sets participating in the relationship
    b. Types
        i. Unary



1.
2. A person married to a Person

ii. Binary



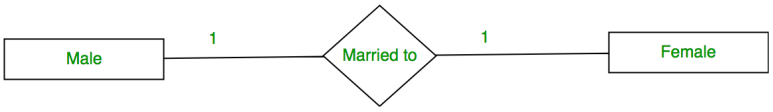
1.
2. A student enrolled in a course

iii. N-ary

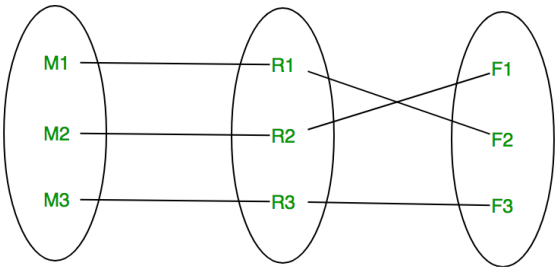
1. N entity sets participate in the relationship

xii. **Cardinality**

1. **One to One**

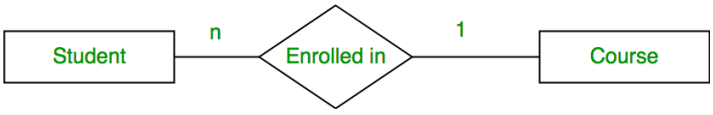


a.



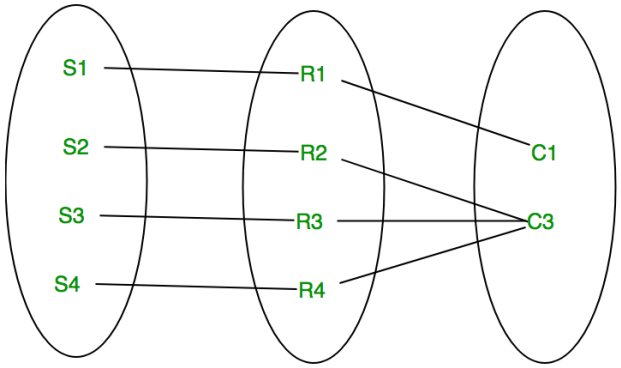
b.

2. **Many to One**



a.

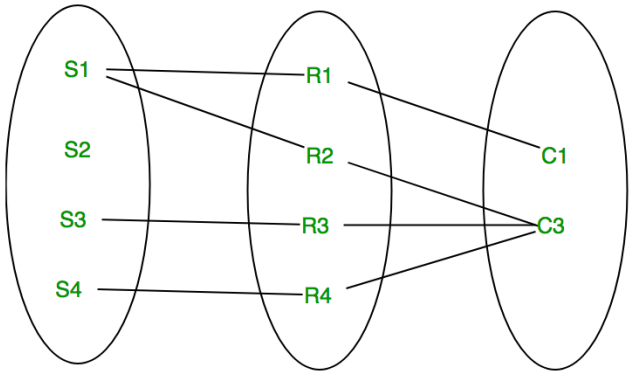
b.



3. **Many to Many**

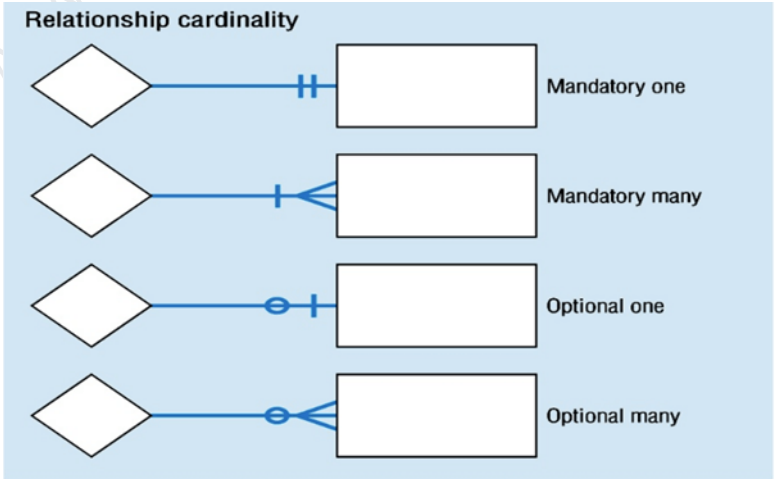
a.

b.



4. Representation in ER diagram

a.

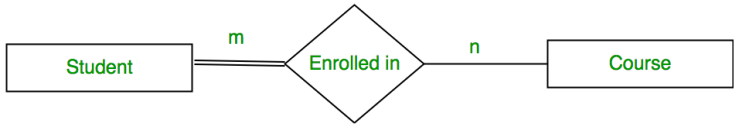


xiii. Participation Constraint

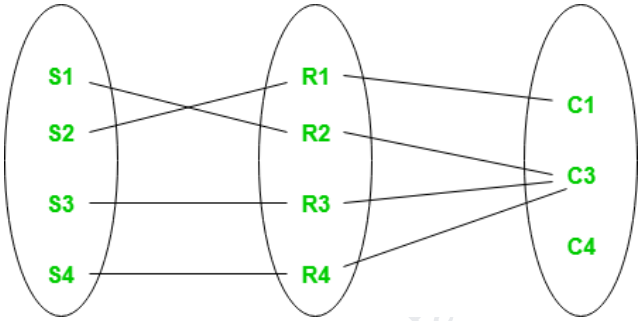
1. **Total participation**
   a. Every student must enroll in at least one course -> total participation of student
   b. Represented using double lines

2. **Partial participation**
    a. The entity set may or may not participate in the relationship
    b. Some courses may not be enrolled by any student
    c. E.g. Total participation of student and partial participation of Course

d.

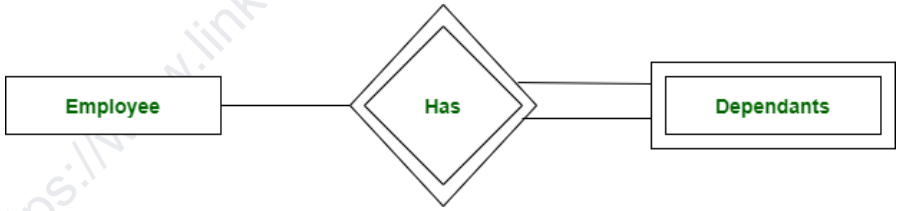


e.



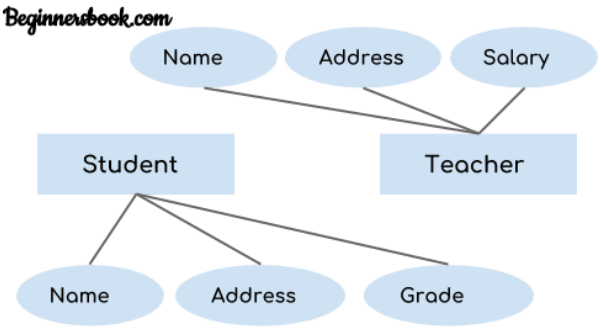
xiv. Weak entity types & Identifying relationships

1. There exists some entity for which key attribute can't be defined
2. There is no existence of Employee-dependants without Employee
3. A weak entity type is represented by a **double rectangle,** and it always has **total participation** in a relationship
4. The relationship between a **weak entity type** and its **identifying strong entity type** is represented by **double diamond**
5.

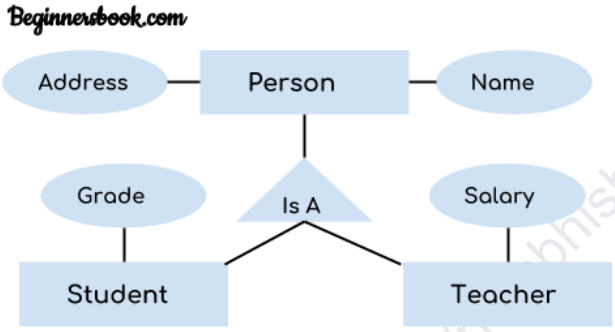


xv. Generalized Entity

1. Uses bottom-up approach
2. Two or more low-level entities combine to form a generalized higher-level entity
3. Ex :

4. 

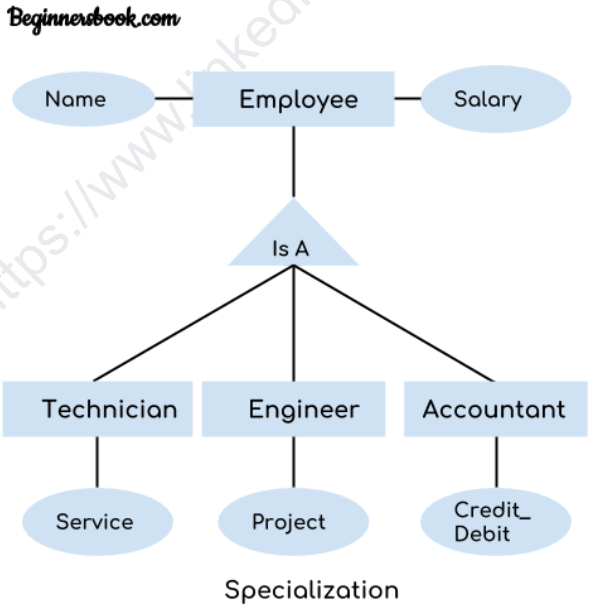
Before Generalization

5. 

Generalization

xvi. Specialization entity

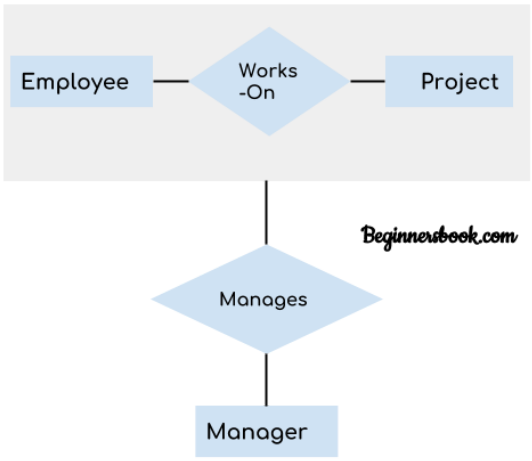
1. Reverse of generalization
2. Entity divided into sub-entities

3. 

Specialization

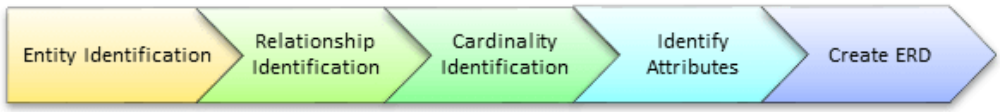
xvii. Aggregation

1. 2 or more entities act as a single entity in a relationship

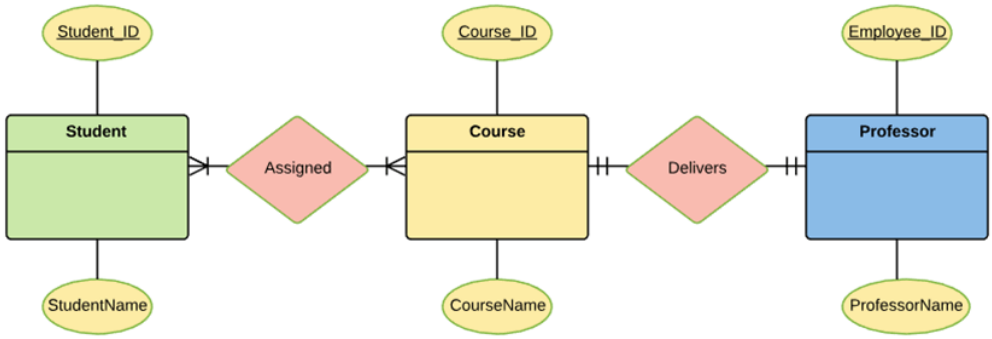


2.
3. A manager manages both employees and the project

xviii. Drawing ER model

1. 

2. Eg.
3. 3 entities: Student, Course, Professor
4. Relationships:
    a. Student is assigned a course
    b. Professor delivers a course
5. Cardinality
    a. A student can be assigned many courses
    b. A professor only delivers a single course
6. Attributes
    a. Student_name, student_id (key)
    b. Course_name, course_id(key)
    c. Prof_name, prof_id(key)
7. Diagram

a. 

7. **Relational Model** (Difference b/w ER model and Relational Model) Read more

	ER Model	**Relational Model**
Used	To describe entities and relationships between them	To represent a collection of tables and relations between them
Type	**High level, Conceptual**	**Implementation/ Representational model**
Components	Entity, Entity Type, Entity Sets	Domain, Attributes, Tuples
Used by	People who don't have programming knowledge	Programmers
Mapping	This model describes the mapping of cardinality	Does not describe cardinality

---

8. All types of Keys in Relational Model (Primary Key, Super Key, Candidate Key, Foreign Key) (Hint: Important)

1. **Keys**
    a. **KEYS in DBMS** is an attribute or set of attributes which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables
    b. Types :
        i. **Primary Key**
            1. It uniquely identifies any tuple of the table, there can be many keys but only 1 primary key
        ii. **Candidate Key**
            1. All candidates for primary key(as strong as the primary key but weren't chosen as primary key)
        iii. **Super Key**
            1. Superset of candidate key
            2. (Name, Roll), Roll, (Age, Aadhar) could be superkeys
        iv. **Foreign Key**
            1. The primary key of some other table used to link 2 tables eg.

DeptId

9. Armstrong Axioms and All Types of Functional Dependencies (Transitive and Relative)

a. Functional Dependency

i. Functional Dependency is a constraint between two sets of attributes in relation to a database. A functional dependency is denoted by an arrow (→). If an attribute A functionally determines B, then it is written as A → B.

ii. A function dependency A → B means for all instances of a particular value of A, there is the same value of B.

iii. Types

1. Trivial FD
    a. X->Y and Y is a subset of X e.g ABC->BC
2. Non Trivial FD
    a. X->Y and Y is not a subset of X e.g. AB->C
        i. Completely non Trivial: X intersection Y = NULL
        ii. Semi non Trivial: X intersection Y != NULL

b. Armstrong's Axioms GFG

- Set of inference rules or axioms, used to test the logical implication of Functional dependencies
- **Axioms**
    - **Axiom of Reflexivity**
        - If B is a subset of A, then A-> B is trivial property
    - **Axiom of Augmentation**
        - A->B then AC->BC
    - **Axiom of Transitivity**
        - A->B and B->C then A->C
- Secondary Rules
    - Union: A->B, A->C => A->BC
    - Composition: A->B, X->y => AX->BY
    - Decomposition A->BC => A->B, A->C
    - Pseudo Transitivity: A->B, BC->D => AC->D

---

10. Database Anomalies (Insert, Delete and Update) GFG

a. Due to redundancy in table

b. Anomalies

i. Example table

1.

id	name	branch	hod
1	a	cse	u
2	b	cse	u
3	c	cse	u
4	d	mech	v

ii. **Insertion Anomaly**
   1. When certain data cannot be inserted into table without insertion of some other data
   2. E.g. you cannot insert data about civil branch until at least 1 student is enrolled in civil

iii. **Deletion Anomaly**
   1. Deletion of some data causes deletion of some wanted data
   2. E.g. You delete tuple with id 4, then you lose info about mech branch

iv. **Modification/Updation Anomaly**
   1. When updating a single data causes updates to all of its copies
   2. E.g. if hod of cse changes then you will have to update everywhere
   3. If for some reason some copies aren't updated then it causes **inconsistency in data** e.g. you won't be able to determine who is hod of cse since it will have different values in different tuples

v. Solution: Normalization

---

11. **Normalization -> What is Normalization**? Why do we need Normalization ?

Database normalization is the process of organizing the attributes of the database to reduce or eliminate data redundancy (having the same data but at different places) .

---

12. All the Types of Normalization, **1NF, 2NF, 3NF, and BCNF**. (Hint: They won't ask you to Convert the Functions into 1NF, 2NF, and All, Just Know the Basics of Each Normal Form & It Would Be Better if you Understand Every NF Form with an Example) (**Important topic**) GFG link | javapoint

1. **1NF**
   a. A relation is in 1NF if it contains an atomic value.

   b. There should be no multi valued attribute
2. **2NF**
   a. Should be in 1NF
   b. **Partial dependency:** a proper subset of candidate key determines a non prime attribute
   c. There should be no **Partial Dependency**
   d. I.e. No non-prime attribute is determined by a subset of Candidate key
3. **3NF**
   a. Should be in 2NF
   b. There should be no **Transitive Dependency**
   c. I.e. No F.D of the form X->Y where both X,Y are non prime
   d. For every Functional Dependency A->B, one of these must hold
      i. A is superkey or
      ii. B is prime attribute(subset of Candidate Key)
4. **BCNF,** Boyce-Codd Normal Form
   a. Should be in 3NF
   b. For every functional dependency A->B
      i. A is super key

---

13. SQL Queries (Learn with one example of Each Query, Not Much, **Best Resource : SQL w3 Resource**) **Practice queries on Leetcode**

---

14. File Structures in Database (Indexing, Sparse Indexing, **B- Tree, B+ trees**) (Source GFG)

Q. Where are indexes stored?

1. **Indexing**
   a. **Primary**
      i. Sorted and Unique
      ii. Index size = number of blocks in Hard Drive
      iii. Cost = logN + search in 1 block
      iv. Sparse index
   b. **Clustered**
      i. Sorted and not unique
      ii. Index same as primary (points to block)
      iii. Cost = logN + search in more than 1 blocks e.g. key = 2 can be present from 5 - 15 blocks
   c. **Secondary**
      i. **Dense index**
      ii. On Key attribute (unsorted):

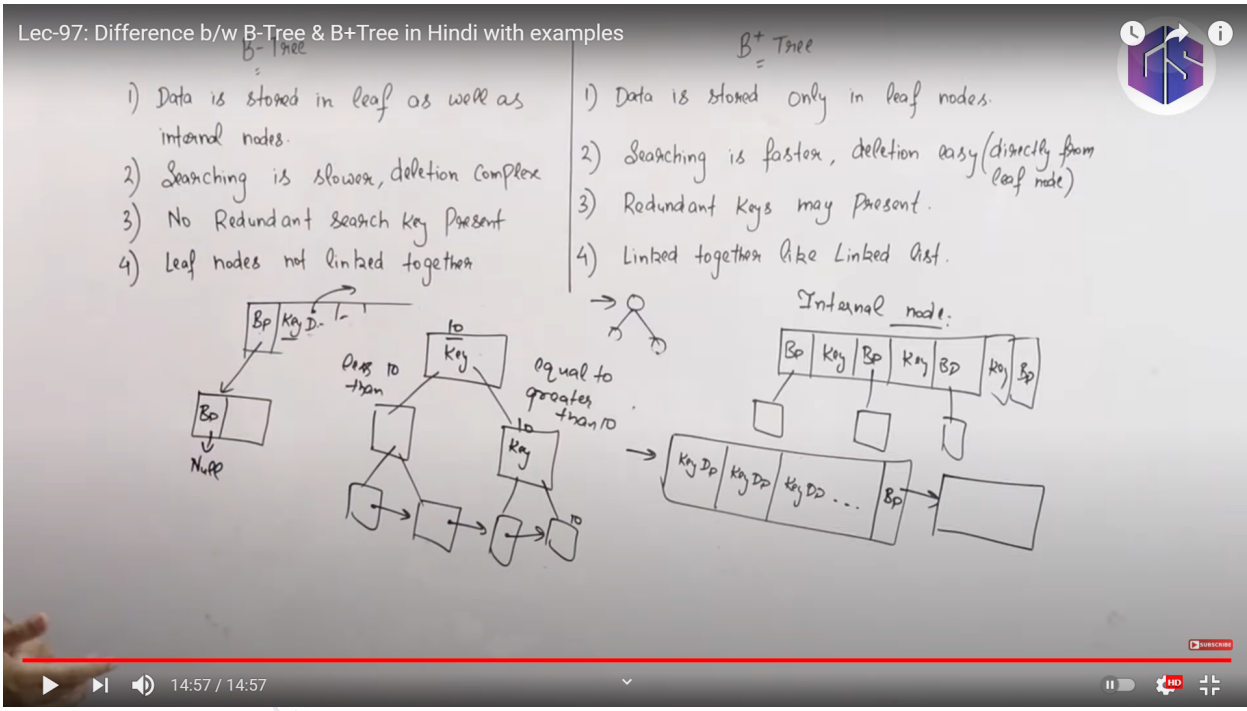
1. Make dense index in sorted order and binary search

iii. On Non-Key attribute (unsorted)

1. Make dense index in sorted order but keep unique values, then each unique value points to an intermediate block containing pointers to a location with that data
2. I.e Index-> Block containing all pointers with given index-> actual record

2. Btree vs B+ tree [YT](#) [Gfg](#)

a.

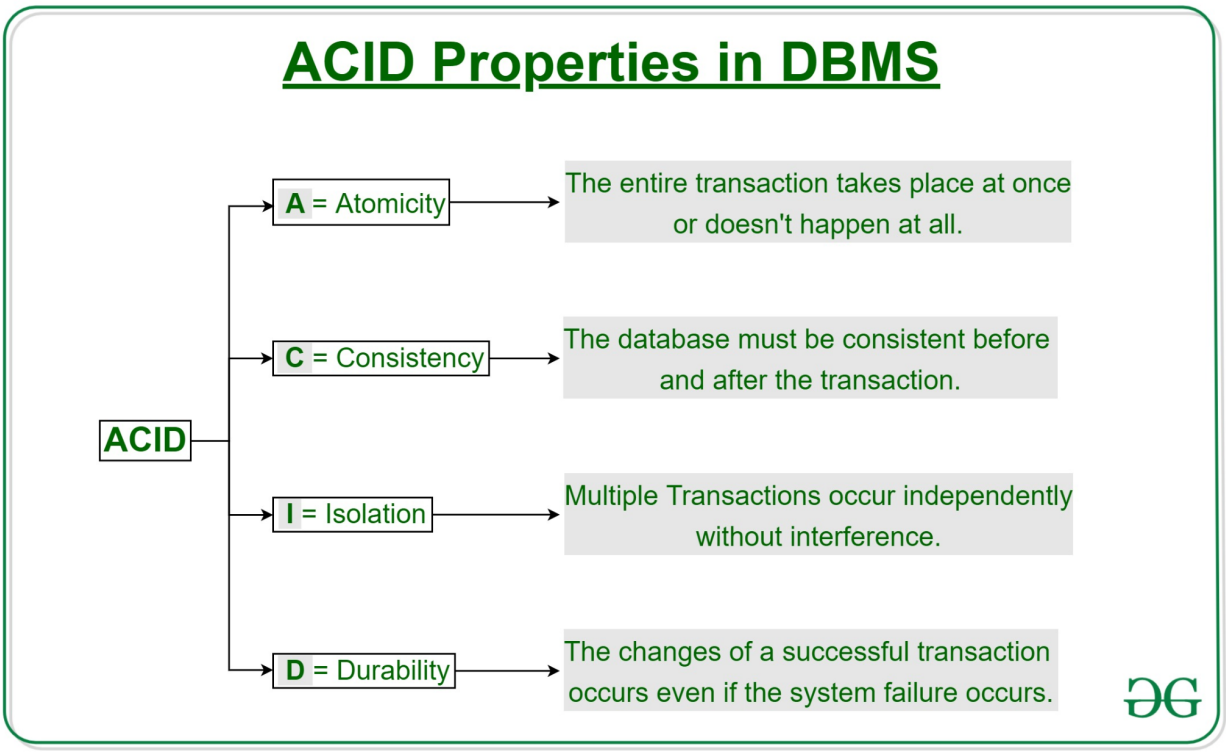


---

15. Transactions, Operations of Transactions, **Transaction's ACID Properties (Important)** [Gfg](#)

1. Transaction

a. A transaction is a single logical unit of work which accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.



b.

c. Atomicity:
   i. Either complete or none
   ii. If some failure occurs in between transactions it is **aborted** and **rolled back**
   iii. After completion, we **commit** and changes are permanent

d. Consistency:
   i. Total money before transaction = total money after the transaction

e. Isolation : isolation levels
   i. 2 transactions don't interfere with each other
   ii. Isolation levels
      1. Read uncommitted
      2. Read committed
      3. Repeatable read
      4. Serialize

f. Durability:
   i. Changes persist.

---

16. State of Transactions i.e Committed, partially Committed

1. To avoid dirty data due to failures, transactions support 2 operations
    a. **Commit**
        i. When all operations of a transaction are done, only then the database gets updated, until then all operations are done on buffer
    b. **Rollback**
        i. In case of failure, the changes are not committed and get rolled back

---

17. **Concurrency Control**, Lock and all over Transactions GfG

1. **Schedule:**
    a. When 2 or more transactions are taking place at the same time, depending on the order of operations, we might have **inconsistency** in the database
    b. To solve this, we define an order of operations called schedule
2. 2 types of schedule
    a. **Serial schedule**
        i. T2 takes place only after T1 has completely finished
        ii. Always consistent
    b. **Concurrent Schedule**
        i. T1 and T2 take place at the same time, we need to **Serialize** the schedule to avoid inconsistency
3. Conflict serializability GfG
    a. If we can order the schedule in such a way that guarantees consistency => conflict serializable schedule