**Data Base Management System**

[I. Introduction to DBMS 4](#_Toc123505614)

[1. Data 4](#_Toc123505615)

[2. Information 4](#_Toc123505616)

[3. Database 4](#_Toc123505617)

[4. Database Management System 4](#_Toc123505618)

[5. Need for DBMS 4](#_Toc123505619)

[II. Data Models and Schema 4](#_Toc123505620)

[1. Data Model 4](#_Toc123505621)

[a. Conceptual Data Model 4](#_Toc123505622)

[b. Representational Data Model 4](#_Toc123505623)

[c. Physical Data Model 4](#_Toc123505624)

[2. Schema 4](#_Toc123505625)

[3. Instance 5](#_Toc123505626)

[III. ER Model 5](#_Toc123505627)

[1. Entity 5](#_Toc123505628)

[a. Types 5](#_Toc123505629)

[2. Attribute 6](#_Toc123505630)

[a. Based on Composition 7](#_Toc123505631)

[b. Based on count of values that can be stored 7](#_Toc123505632)

[c. Based on how the attribute value can be stored 8](#_Toc123505633)

[d. Based on keys 8](#_Toc123505634)

[3. Relationships 8](#_Toc123505635)

[a. Strong Relationship 9](#_Toc123505636)

[b. Weak Relationship 9](#_Toc123505637)

[c. Degree of Relationship 9](#_Toc123505638)

[d. Cardinality 9](#_Toc123505639)

[e. Specialization 12](#_Toc123505640)

[f. Generalization 12](#_Toc123505641)

[g. Aggregation 13](#_Toc123505642)

[IV. Relational Model 13](#_Toc123505643)

[1. Relational model keys 13](#_Toc123505644)

[a. Super Key 14](#_Toc123505645)

[b. Primary Key 14](#_Toc123505646)

[c. Candidate Key 14](#_Toc123505647)

[d. Alternate Key 15](#_Toc123505648)

[e. Foreign key 15](#_Toc123505649)

[f. Surrogate Key 15](#_Toc123505650)

[2. Integrity Constraints 15](#_Toc123505651)

[a. Domain Constraints 15](#_Toc123505652)

[b. Entity Constraints 16](#_Toc123505653)

[c. Referential Constraints 16](#_Toc123505654)

[d. Key Constraints 16](#_Toc123505655)

[3. Relational Algebra 16](#_Toc123505656)

[V. Transaction Control Language 16](#_Toc123505657)

[1. COMMIT 17](#_Toc123505658)

[2. SAVEPOINT 18](#_Toc123505659)

[3. ROLLBACK 18](#_Toc123505660)

[4. Locks 19](#_Toc123505661)

[a. READ LOCK 19](#_Toc123505662)

[b. WRITE LOCK 19](#_Toc123505663)

[VI. Normalization 19](#_Toc123505664)

[1. Functional Dependency 19](#_Toc123505665)

[a. Armstrong’s axioms/properties of functional dependencies 20](#_Toc123505666)

[b. Types of Functional Dependencies 21](#_Toc123505667)

[2. Anomolies 21](#_Toc123505668)

[a. Insertion Anomaly 21](#_Toc123505669)

[b. Update Anomaly 21](#_Toc123505670)

[c. Delete Anamoly 22](#_Toc123505671)

[3. Normalization 22](#_Toc123505672)

[a. First Normal Form (1NF) 22](#_Toc123505673)

[b. Second Normal Form (2NF) 23](#_Toc123505674)

[c. Third Normal Form (3 NF) 24](#_Toc123505675)

[d. Boyce-Codd Normal Form (BCNF) 25](#_Toc123505676)

[VII. Indexing 27](#_Toc123505677)

[1. Primary Index 28](#_Toc123505678)

[a. Dense index 28](#_Toc123505679)

[b. Sparse index 28](#_Toc123505680)

[2. Clustering Index 28](#_Toc123505681)

[3. Secondary Index 29](#_Toc123505682)

[4. Usecases of Indexing 30](#_Toc123505683)

[5. Drawbacks of Indexing 30](#_Toc123505684)

# Introduction to DBMS

## Data

Any raw and unprocessed fact that we can record is known as data.

Example - New Delhi, India

## Information

When we process the data to get meaningful facts.

Example - New Delhi is the capital of India.

## Database

A collection of related data organised in a way that data can be easily accessed, managed and updated.

Database can contain the both structured and unstructed data. Relational databases contain only structured data (MySQL, MS SQL) whereas NoSQL (MongoDB) can contain the unstructured data.

## Database Management System

DBMS provides users with an interface or a tool, to perform various operations like creating a database, storing data in it, updating data, creating tables in the database.

Example - MySQL, PostgreSQL, Microsoft Access, Oracle, MongoDB, Cassandra.

In DBMS we have users who write queries using some query language for example SQL (Structured Query Language).

## Need for DBMS

Before DBMS we were using the file system approach.

But it has drwabacks like redundancy, inconsistency, back up and recovery, unauthorized access.

# Data Models and Schema

## Data Model

It describes how data is stored, connected, accessed and updated.

It is classified into 3 types:

### Conceptual Data Model

Describes the database at very high level through the E-R diagrams

### Representational Data Model

It focuses on the design part (or logic) of the database. (Datatypes, ranges etc)

### Physical Data Model

Basically, it deals with physically how the data is stored in the device - like disk, drive, or tapes.

## Schema

A database schema is like a blueprint it doesn’t hold the data itself, but instead it describes the data.

Difference between Database and Database Schema:

|  |  |
| --- | --- |
| Database Schema | Database |
| Does not contain any data of its own | Contain the data |
| Schema structure does not change | Data changes |

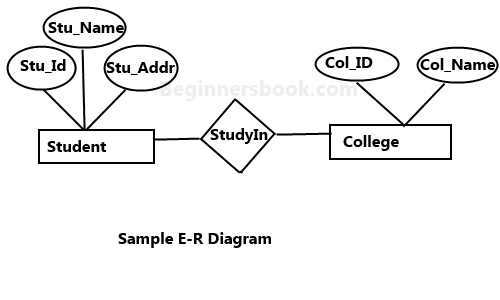
## Instance

Instance of a database is defined as the data or collection of information stored in a database at a particular moment of time.

It contains a snapshot of the database.

# ER Model

Entity Relationship Model is the graphical representation of database design. It is a high level data model that describes data in terms of entities, attributes and relationships.



## Entity

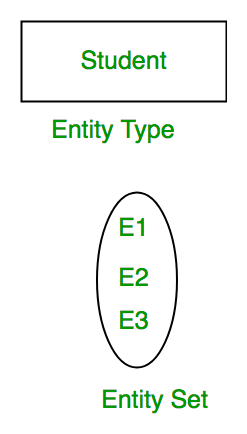
It is a real-world object, class, person or place. Entities can be uniquely identified. It has a physical existence.

It is represented by a rectangle.

### Types

#### Entity Set

Set of entities of the same type at a particular point of time is known as the entity set.



#### Strong Entity

An entity which is not dependent on any other entity.

Strong entity will always have a primary key. It is represented with a rectangle.

#### Weak Entity

A weak entity is usually dependent on a strong entity to ensure its existence.

It does not have any primary key rather contains a discriminator or a partial key to differentiate between the records present in the weak entity set table.

It is represented with a double rectangle. It needs to have participation with strong entity.

Example :

A company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents don’t have existed without the employee. So Dependent will be a weak entity type and Employee will be Identifying Entity type for Dependent.

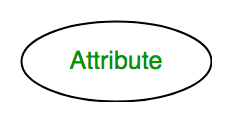


## Attribute

Attributes are the properties that define the entity.

For example, Roll\_No, Name, DOB, Age, Address, Mobile\_No are the attributes that define entity type Student.

In ER diagram, the attribute is represented by an oval.



Types of Attributes

### Based on Composition

#### Simple Attribute

If an attribute cannot be divided further

#### Composite Attribute

If any attribute can be divided into further parts



### Based on count of values that can be stored

#### Single Valued Attribute

If a particular attribute has only one value then it is known as a single valued attribute.

#### Multi Valued Attribute

If an attribute has more than one value then it is known as a multivalued attribute. It is represented by a double oval.

Phone\_No (can be more than one for a given student). In ER diagram, a multivalued attribute is represented by a double oval.



### Based on how the attribute value can be stored

#### Stored Attribute

The initial information which we save in our database is known as a stored attribute.

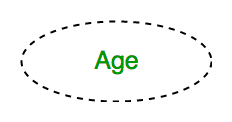
DOB is a stored attribute.

#### Derived Attribute

The value which we can derive from stored attributes.

In ER diagram, the derived attribute is represented by a dashed oval.

Example: Age is a derived attribute because it can be derived from DOB.

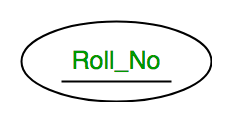


### Based on keys

#### Key Attribute

This is an attribute or set of attributes that can uniquely identify each entity in the entity set.

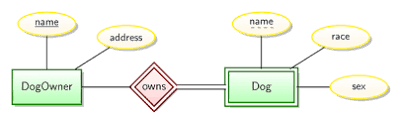
In ER diagram, key attribute is represented by an oval with underlying lines.



#### Foreign Key Attribute

Whenever there is some relationship between two entities there must be some common attribute between them. This common attribute must be the primary key of an entity set and will become foreign key of another entity set

Here name is foreign key in Dog entity which is a primary key of DogOwner



## Relationships

It defines how two or more entities are connected with each other. It is an association among two or more entities. It is represented by a diamond operator in the ER diagram.



### Strong Relationship

The relationship between two independent entities is a strong relationship.

### Weak Relationship

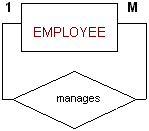
When a relationship exists between a weak entity and its owner entity, then it is called a weak relationship.

### Degree of Relationship

Number of entities participating in a single relationship

#### Unary Relationship

It exists when only one entity participates in a relationship.



Example

An employee could be a manager also, so the manager manages other employees, and all types of employees are in a single entity.

#### Binary Relationship

When two entities participate in a relationship then it is known as binary relationship.



#### n-ary Relationship

When there are n entities set participating in a relation, the relationship is called as n-ary relationship.

### Cardinality

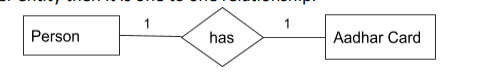
In binary relationship cardinality ratio represents how many times the entity/instance of an entity set can participate in a relationship.

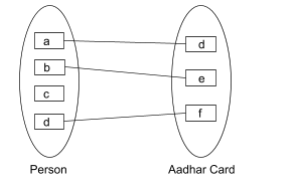
#### One to One

If only one instance of an entity is associated with only one instance of another entity then it is one to one relationship.

Example

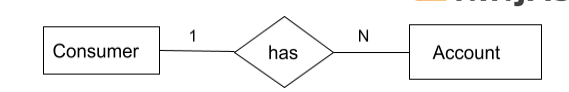
one person could only have one Aadhar Card,



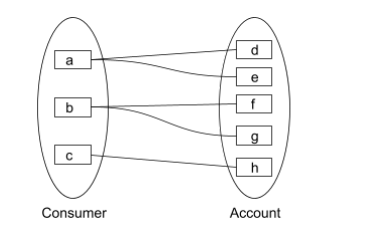


#### One to Many

If a single instance of an entity is associated with more than one instance of another entity.

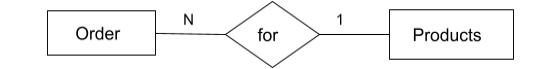


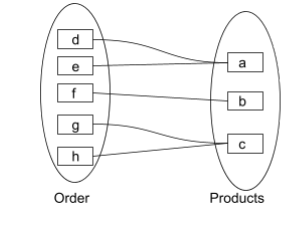
In the above example one customer can have multiple bank accounts.



#### Many to One

If more than one instance of an entity is associated with a single instance of another entity



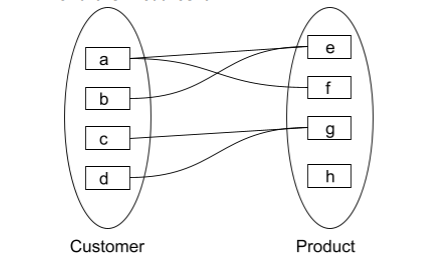


one product could have multiple orders

#### Many to Many

If more than one instance of an entity set is associated with more than one instance of another entity.





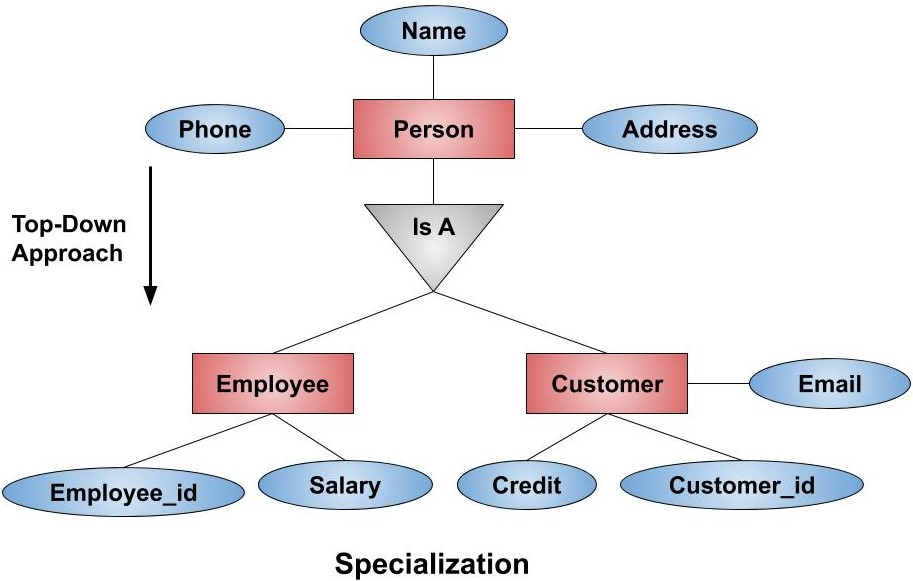
one customer can place many orders and one product can be ordered by multiple customers.

### Specialization

It is a process of specializing an entity type into a more specified entity.

In this process, we specialize a higher-level entity type by adding some additional attributes to the entity type.

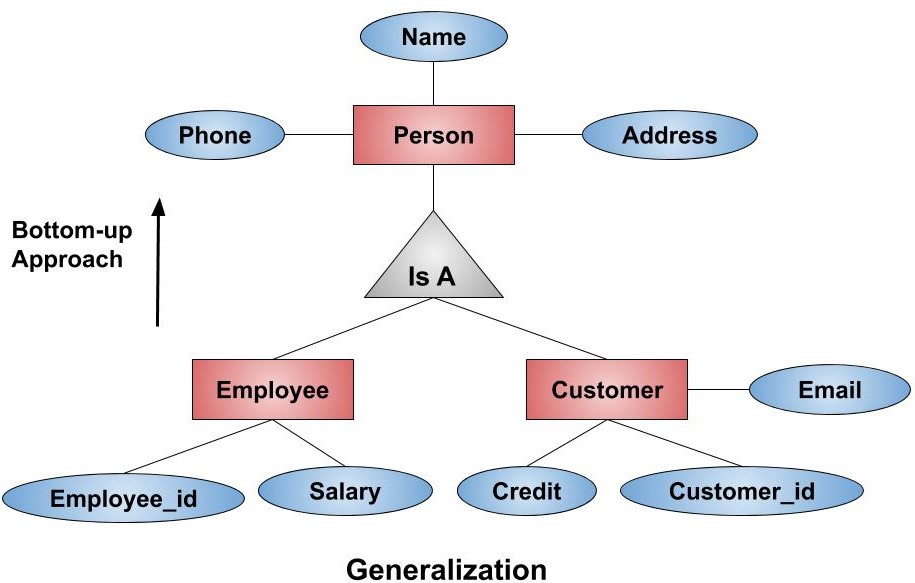
It is a top-down approach in which a higher-level entity is broken into smaller entities.



### Generalization

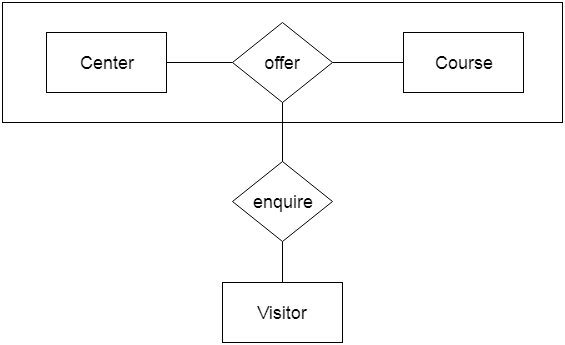
It is a process of generalizing two or more lower-level entity types into a higher-level entity type. Entities are clubbed or grouped together to represent a more generalized view.

It is a bottom-up approach . It is the reverse process of Specialization.



### Aggregation

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher-level entity.



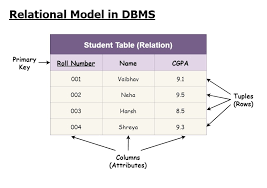
Example

Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.

# Relational Model

The relational model represents how data is stored in Relational Databases. A relational database stores data in the form of relations (tables).

STUDENT



## Relational model keys

### Super Key

A superkey is a combination of columns that uniquely identifies any row within a relation/table.

It is a superset of all the key. It can have single attribute or multiple attributes which can be combined together as a unique set.

If a STUDENT relation has [stud\_id, stud\_name , stud\_address,stud\_phone]

{ stud\_id }

{ stud\_id , stud\_name }

{ stud\_id, stud\_name , stud\_address}

{ stud\_id, stud\_name , stud\_address,stud\_phone }

Etc.

So any permutation and combination (PNC) of the keys or attributes in a relation or a table can qualifies as a superkey.

### Primary Key

It is a unique identifier which helps us to identify each and every tuple uniquely.

Tuple is any row in the table.

No two rows have the same primary key.

Every row has to have primary key, so primary key can not have a null value.

The primary key in the table cannot be changed.

Ex : stud\_id

#### Composite Key

Primary key which is formed using at least two attributes.

{stud\_name , stud\_address} (this is one of the candidate keys)

#### Compound Key

Primary keys which are formed using two foreign keys which have been referenced in some other tables/relations.

Let us say STUDENT relation has [stud\_id, stud\_name , stud\_address,stud\_phone]

COURSE relation, it has [course\_id, course\_name ]

Here STUDENT does not have COURSE id and vice-versa.

Let us take

ENROLLMENT [stud\_id, course\_id,enrollment\_fee]

If in ENROLLMENT we make { stud\_id, course\_id } as primary key it will become Compound key.

### Candidate Key

The minimum set of attributes that can uniquely identify each tuple.

Candidate key can have a collection of more than one attribute to form a key.

Primary key is selected out of Candidate key.

Candidate Key value should not be null.

Example :

{stud\_id}

{stud\_name , stud\_address}

can be the Candidate keys.

### Alternate Key

All the candidate keys except the primary key are known as alternate keys.

### Foreign key

It creates the relationship between two tables.

It helps us to cross refernce between two tables.

It is also used to maintain data integrity consistency.

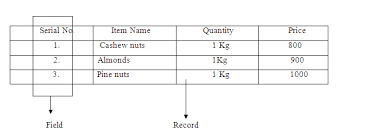
If a STUDENT relation has [stud\_id, stud\_name , stud\_address,stud\_phone]

Let’s take COURSE relation, it has [course\_id, stud\_id,course\_name ] , here stud\_id is the foreign key referencing to STUDENT relation.

### Surrogate Key

To uniquely identify the data, generally we give a surrogate key as an integer.

Surrogate key also called a synthetic primary key, is generated when a new record is inserted into a table automatically by a database.



## Integrity Constraints

Integrity constraints are rules that should be applied on database columns to ensure the validity of data.

With all the authorised changes to the database happening, integrity constraints ensure that there is no loss of data consistency.

There are three type of integrity constraints – 1. Domain Constraints 2. Entity Constraints 3. Referential Constraints

### Domain Constraints

The domain integrity constraints restrict the value in the particular attributes of a relation.

Example: Let us say we need to ensure that semester student is registering for is one of fall, winter, spring or summer.

**CREATE** **TABLE** **section** **(**

course\_id VARCHAR**(**8**),**

sec\_id VARCHAR**(**8**),**

semester VARCHAR**(**6**),**

year NUMERIC**(**4**,**0**),**

building VARCHAR**(**15**),**

room\_number VARCHAR**(**7**),**

time\_slot\_id VARCHAR**(**4**),**

**primary** **key** **(**course\_id**,** sec\_id**,** semester**,** year**),**

**check** **(**semester **in** **(**’Fall’**,** ’Winter’**,** ’Spring’**,** ’Summer’**))**

**);**

### Entity Constraints

It puts constraints on Primary key i.e. primary should be unique and does not have NULL value.

### Referential Constraints

It is specified between two relations and helps maintain consistency among the tuples of two relations.

This constriant is applied on the foreign key.

Foreign Key must be having the matching primary key or it must be NULL.

Rules

#### Insert Constraint

We can not insert a record in the **child/referencing** relation, If the corresponding record does not exist in the **master/ referenced** relation.

#### Delete Constraint

We can not delete the record from **master/ referenced** relation if the corresponding record lying in the **child/referencing** relation.

### Key Constraints

* Not Null
* Unique
* Default
* Check
* Primarykey
* Foreignkey

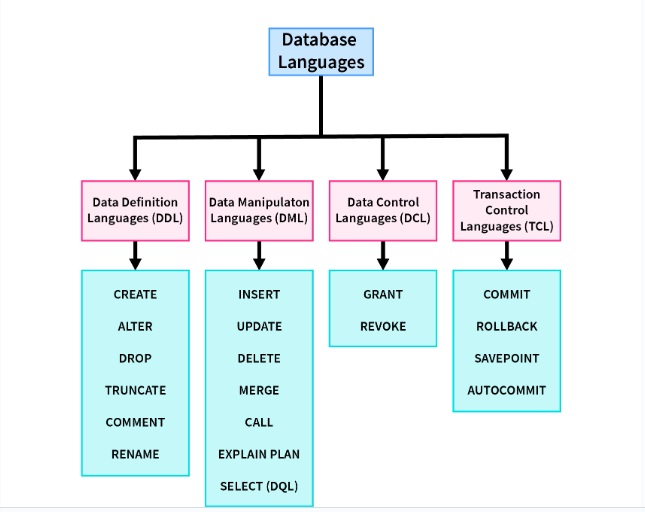
## Relational Algebra

* Selection (σ)
* Projection (π)
* Union (U)
* Intersection (∩)
* Set Difference (–)
* Cartesian Product (X)
* Rename (ρ) (alias)

# Transaction Control Language

Transaction

Logical unit of work , like mutiple correlated SQL statements also call it automic unit.



## COMMIT

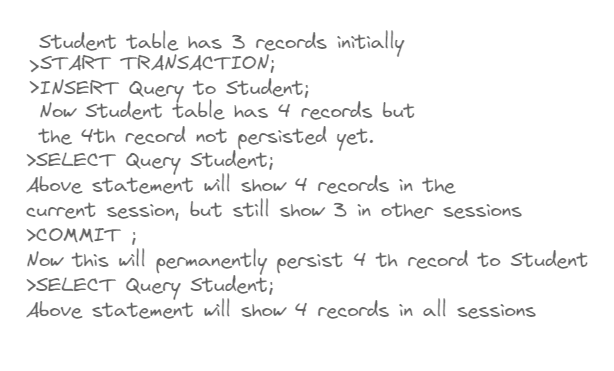
It saves all the work done.

It ends the current transaction and makes permanent changes during the transaction.

Some DBMS softwares by default set to AUTO COMMIT mode,we can change that setting .

>SET autocommit=OFF;

Consider the below sequence , COMMIT will save all the work done from START TRANSACTION till the end.



## SAVEPOINT

This command saves the current state into save point. That save point can later be accessed.

>SAVEPOINT <savepoint\_name>;

## ROLLBACK

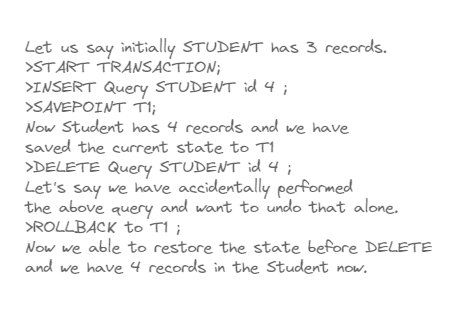
This commands uncommit all the changes or restore the current state to any previously saved state.

>ROLLBACK;

It restores the database to last committed state.

>ROLLBACK TO SAVEPOINT <savepoint\_name>;

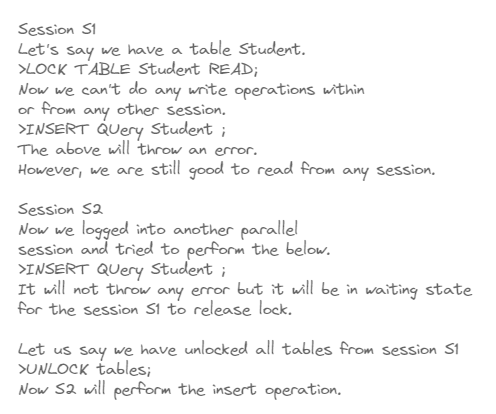
It restores the database to a particular savepoint state.



## Locks

### READ LOCK

This lock allows a user to only read the data from a table.



### WRITE LOCK

* The only session which holds the lock of the table can read and write data both from the table.
* All other sessions cannot access the data of the table not even able to read until the WRITE lock is released from the holding session.

To get the list of active processes we can use the below command.

>SHOW PROCESSLIST;

# Normalization

## Functional Dependency

A functional dependency (FD) is a relationship between two attributes, typically between the primarykey and other non-key attributes within a table.

It is denoted as **X → Y.**

The attribute set on the left side of the arrow, X is called **Determinant**, while on the right side, Y is called the **Dependent**.

i.e X determines the value of Y , and Y value is dependent on X.

A functional dependency X->Y in a relation holds if two tuples having same value of attribute X also have same value for attribute Y.

Let us take relation R1 below

|  |  |
| --- | --- |
| C1 | C2 |
| 1 | 1 |
| 2 | 4 |
| 3 | 9 |

C1 -> C2 holds here and also C2->C1 holds good too, as both C1 and C2 are having unique values.

Let us take relation R2 below

|  |  |
| --- | --- |
| C1 | C2 |
| 1 | 1 |
| 2 | 4 |
| 3 | 9 |
| 4 | 9 |

Here only C1->C2 holds as C1 has unique values , C2->C1 does not hold as C2 have two values same (9) but corresponding values in C1 are different(3 , 4 ).

|  |  |
| --- | --- |
| C1 | C2 |
| 1 | 1 |
| 2 | 4 |
| 3 | 9 |
| 3 | 9 |

Here both C1->C2 and C2->C1 holds good.

### Armstrong’s axioms/properties of functional dependencies

#### Reflexivity

If Y is a subset of X, then X→Y holds by reflexivity rule

For example, {roll\_no, name} → name is valid.

#### Augmentation

If X → Y is a valid dependency, then XZ → YZ is also valid by the augmentation rule.

For example, If {roll\_no, name} → dept\_building is valid, hence {roll\_no, name, dept\_name} → {dept\_building, dept\_name} is also valid.

#### Transitivity

If X → Y and Y → Z are both valid dependencies, then X→Z is also valid by the Transitivity rule.

For example, roll\_no → dept\_name & dept\_name → dept\_building, then roll\_no → dept\_building is also valid.

### Types of Functional Dependencies

#### Trivial Functional Dependency

If a dependent is always a subset of the determinant.

i.e If A→ B and B⊆A

#### Non-trivial Functional Dependency

If the dependent is strictly not a subset of the determinant.

i.e If A→ B and A ∩ B = NULL

#### Multivalued Functional Dependency

If the attributes of the dependent set are not dependent on each other.

i.e. If a → {b, c} and there exists no functional dependency between b and c.

#### Transitive Functional Dependency

If a → b & b → c, then according to axiom of transitivity, a → c.

## Anomolies

A database anomaly is an inconsistency in the data resulting from an operation like an update, insertion, or deletion.

There can be inconsistencies when a record is held in multiple places(redundancy) and not all of the copies are updated.

### Insertion Anomaly

When certain attributes cannot be inserted into the database without the presence of other attributes.

| **stu\_id** | **stu\_name** | **stu\_address** | **stu\_club** |
| --- | --- | --- | --- |
| 220 | Annamalai | Kerala | yoga |
| 220 | Muthu | Kerala | Music |
| 231 | Mukesh | Mumbai | Crypto |
| 232 | Muni | Karnataka | Public Speaking |
| 232 | Muni | Karnataka | Arts |

For example, in the above table if a new student named Nanda has joined the college and he has not enrolled in any club yet, we can't insert the data of Nanda into the table since the st\_club field cannot accept null values.

### Update Anomaly

when one or more instances of duplicated data are updated but not all.

| **stu\_id** | **stu\_name** | **stu\_address** | **stu\_club** |
| --- | --- | --- | --- |
| 220 | Annamalai | Kerala | yoga |
| 220 | Muthu | Kerala | Music |
| 231 | Mukesh | Mumbai | Crypto |
| 232 | Muni | Karnataka | Public Speaking |
| 232 | Muni | Karnataka | Arts |

For student Muni as he belongs to two clubs at the college. If we want to change Muni's address, we must update it twice otherwise the data will be inconsistent.

### Delete Anamoly

When certain attributes are lost due to the deletion of other attributes.

the college at some point closes the club crypto, then deleting the rows that contain s\_club as crypto would also delete the information of student Mukesh since he belongs only to this department.

| **stu\_id** | **stu\_name** | **stu\_address** | **stu\_club** |
| --- | --- | --- | --- |
| 220 | Annamalai | Kerala | yoga |
| 220 | Muthu | Kerala | Music |
| 231 | Mukesh | Mumbai | Crypto |
| 232 | Muni | Karnataka | Public Speaking |
| 232 | Muni | Karnataka | Arts |

## Normalization

Normalization is the process of organizing the data by minimizing the redundancy from a relation or set of relations.

It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.

Normalization divides the larger table into smaller and links them using relationships.

### First Normal Form (1NF)

For a Relation/table to justify 1NF it needs to satisfy 3 basic conditions

● Each attribute should contain atomic values. (i.e. No multivalued attributes)

● Each value stored in an attribute should be of the same type.

● All the attributes in a table should have unique names.

Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP\_PHONE.

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385, 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389, 8589830302 | Punjab |

The decomposition of the EMPLOYEE table into 1NF has been shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385 | UP |
| 14 | John | 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389 | Punjab |
| 12 | Sam | 8589830302 | Punjab |

### Second Normal Form (2NF)

It should be in First Normal Form.

It should not have any partial dependencies.

|  |  |  |  |
| --- | --- | --- | --- |
| Course\_Id | Stud\_Id | Marks | Course\_Name |
| CS101 | 1 | 10 | C Progamming |
| CS102 | 1 | 9 | Maths-I |
| CS103 | 1 | 8 | Maths-II |
| CS101 | 2 | 9 | C Progamming |
| CS102 | 2 | 8 | Maths-I |

From the above table, if we need to know how much marks for Stud\_Id got Stud\_Id is not alone enough, we need the Course\_Id also to determine in which course how many marks he has got.

So { Course\_Id , Stud\_Id } -> Marks

So here the primary key combination of { Course\_Id , Stud\_Id }.

Now if we need to know Course\_Name for a particular Course\_Id , the attribute Stud\_Id is not required.

So Course\_Id -> Course\_Name

**Partial Dependency** :If one part of a composite primary key(we call it candidate key also) can independently determine another attribute.

The above table have partial dependency Course\_Id -> Course\_Name.

So the decomposition of the above table is as below (just take out the partial dependency from that in another table)

|  |  |  |
| --- | --- | --- |
| Course\_Id | Stud\_Id | Marks |
| CS101 | 1 | 10 |
| CS102 | 1 | 9 |
| CS103 | 1 | 8 |
| CS101 | 2 | 9 |
| CS102 | 2 | 8 |

|  |  |
| --- | --- |
| Course\_Id | Course\_Name |
| CS101 | C Progamming |
| CS102 | Maths-I |
| CS103 | Maths-II |

### Third Normal Form (3 NF)

It should already be in the 2NF.

It should not have Transitive Dependency

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

* X is a super key.
* Y is a prime attribute, i.e., each element of Y is part of some candidate key. (no non-primary-key attribute is transitively dependent on the primary key)

So If an attribute is dependent on non-prime attribute then there might be transitive dependency.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Name | State | Country |
| 1 | Ram | Andhra | India |
| 2 | Rahul | Texas | USA |

Let us see if above table has transitive dependency.

We can determine State from ID as ID is primary key.

ID 🡪 State

We can also determine Country from State.

State 🡪 Country

Now we should have ID 🡪 Country, in otherwords Country is a non-primary attribute which is transitively dependent on the primary attribue ID.

How to decompose? as we have the problem with Country we can take this out.

|  |  |  |
| --- | --- | --- |
| ID | Name | State |
| 1 | Ram | Andhra |
| 2 | Rahul | Texas |

|  |  |
| --- | --- |
| State | Country |
| Andhra | India |
| Texas | USA |

### Boyce-Codd Normal Form (BCNF)

It is an extension of 3NF and is also known as 3.5 Normal Form.

For a table to be in the Boyce-Codd Normal Form, it should satisfy 2 rules:

* It should be in the Third Normal Form.
* For every functional dependency X → Y, X is the super key of the table.

So it is stricter than 3NF.

A prime attribute shouldn’t be dependent on /derived from a non-prime attribute, so LHS should be always superkey.

|  |  |  |
| --- | --- | --- |
| Student | Course | Tutor |
| 101 | Java | Raj |
| 101 | Python | Ram |
| 102 | Java | Raj |
| 102 | Python | Raghu |

Example for remembering purpose Student 🡪 2 repeating after each [{101 , 101 } {102 , 102 }]

For Course 🡪 consecutively repeating {Java , Python}

For Tutor-> consecutively repeating just change the last value to Raghu [Raj , Ram , Raj , Raghu]

Here the primary key is composite of {Student, Course}

We have the following functional dependencies

{Student, Course} 🡪 {Tutor}

Tutor 🡪 Course (this is violating the BCNF as Tutor is not a superkey.)

To decompose the relation based on BCNF, We just need to use to link decomposed tables Tutor which has problem.

|  |  |
| --- | --- |
| Student | Tutor |
| 101 | Raj |
| 101 | Ram |
| 102 | Raj |
| 102 | Raghu |

|  |  |
| --- | --- |
| Course | Tutor |
| Java | Raj |
| Python | Ram |
| Python | Raghu |

Or also we can maintain a Tutor\_Id to link both the decomposed tables.

|  |  |
| --- | --- |
| Student | Tutor\_Id |
| 101 | 1 |
| 101 | 2 |
| 102 | 1 |
| 102 | 3 |

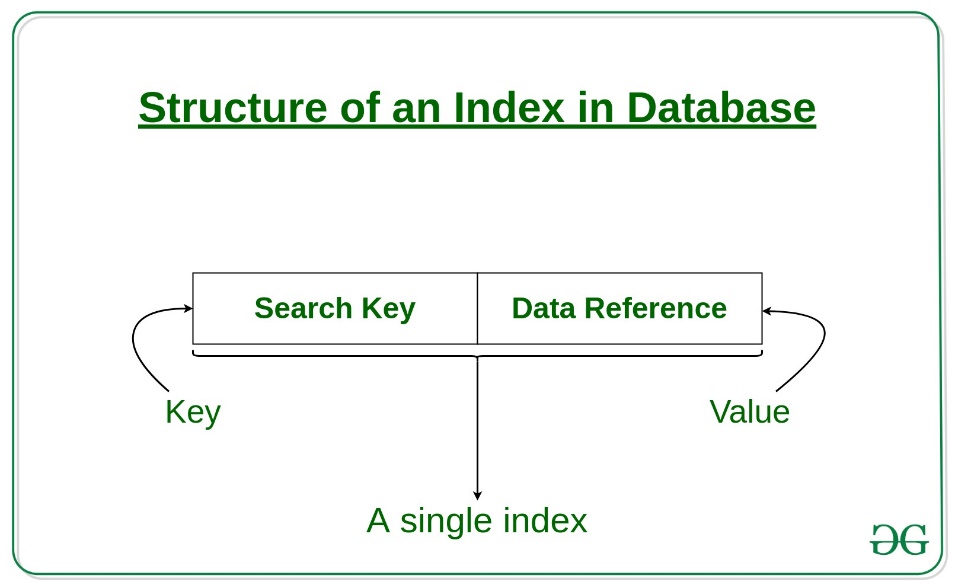
|  |  |  |
| --- | --- | --- |
| Tutor\_Id | Course | Tutor |
| 1 | Java | Raj |
| 2 | Python | Ram |
| 3 | Python | Raghu |

# Indexing

Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.

The index is a type of data structure. It is used to locate and access the data in a database table quickly.

Indexes can be created using some database columns.



## Primary Index

It is the index that is created and ordered on the basis of the primary key of the table.

As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.

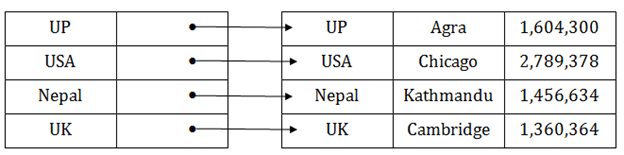
The primary index can be classified into two types: Dense index and Sparse index.

### Dense index

The dense index contains an index record for every search key value in the data file. It makes searching faster.

In this, the number of records in the index table is same as the number of records in the main table.

It needs more space to store index record itself.



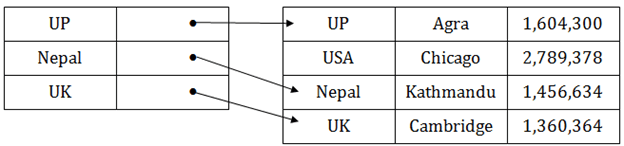
### Sparse index

The index record appears only for a few items in the data file. Each item points to a block as shown.

To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.

We start at that record pointed to by the index record, and proceed along with the pointers in the file (that is, sequentially) until we find the desired record.

Number of Accesses required=log₂(n)+1, (here n=number of blocks acquired by index file)



## Clustering Index

Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

Search key

Here the search key is not a primary key, it is not even a unique identifier or a part of candiadate key.

Data reference

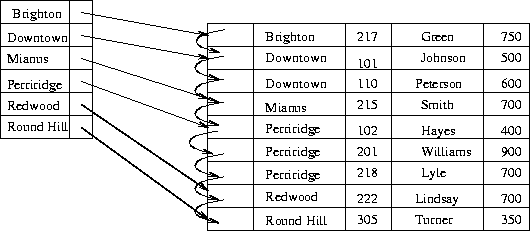
It is not a pointer.

It is a collection of other attributes, which can uniquely identify the tuple.

Here is how the index will look like

|  |  |
| --- | --- |
| Non Primary key | other attributes, which can uniquely identify the tuple |

The index is created on non-primary key columns which may not be unique for each record. In such cases, in order to identify the records faster, we will group two or more columns together to get the unique values and create index out of them.



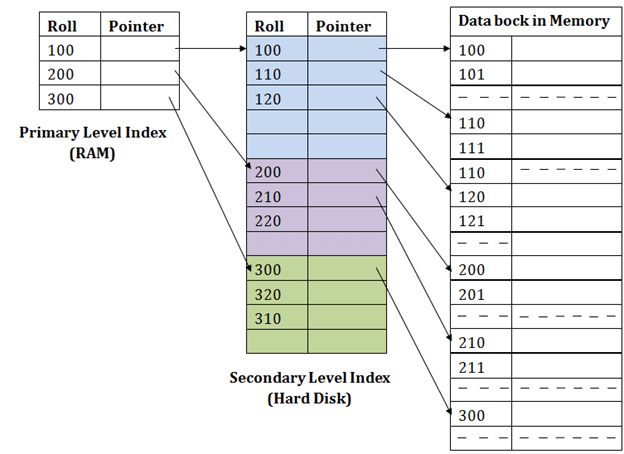
## Secondary Index

It is introduced to tackle the problem when the size of datafile increases i.e. size of table increases, (even the sparse indexing starts to slow down).

Now to overcome this we introduce another level of indexing.

In this, we select a huge chunk of columns initially and put it up at the first level of indexing that is Primary Level index, it is stored in primary memory.

Now each chunk is divided into smaller ranges in the second level of indexing, this is stored in the secondary memory along with the actual datafile.



If you want to find the record of roll 111 in the diagram, then it will search the highest entry which is smaller than or equal to 111 in the first level index. It will get 100 at this level.

Then in the second index level, again it does max (111) <= 111 and gets 110. Now using the address 110, it goes to the data block and starts searching each record till it gets 111.

## Usecases of Indexing

* We can access and retrieve data faster.
* Indexing reduces the number of I/O operations needed to be performed for retrieving data.

## Drawbacks of Indexing

* They take up additional space to store the index table.
* Indexing slows down INSERT,UPDATE and DELETE operations.