PAVENDAR BHARATHIDASAN COLLEGE OF ARTS AND SCIENCE

BCA DEGREE EXAMINATION

YEAR: II

SUBJECT: DATABASE SYSTEMS

SUB CODE: 16SCCCA4

Prepared by

K.VANITHA(Assistant Professor of CS)

Introduction to DBMS

Database is a collection of data and Management System is a set of programs to store and retrieve those data. DBMS is a collection of inter-related data and set of programs to store & access those data in an easy and effective manner.

What is the need of DBMS?

Database systems are basically developed for large amount of data. When dealing with huge amount of data, there are two things that require optimization: **Storage of data** and **retrieval of data**.

Storage: The principles of database systems, the data is stored in such a way that it acquires lot less space as the redundant data (duplicate data) has been removed before storage.

Fast Retrieval of data: Along with storing the data in an optimized and systematic manner, it is also important that we retrieve the data quickly when needed. Database systems ensure that the data is retrieved as quickly as possible.

Purpose of Database Systems

The main purpose of database systems is to manage the data. Consider a university that keeps the data of students, teachers, courses, books etc. To manage this data we need to store this data somewhere where we can add new data, delete unused data, update outdated data, retrieve data, to perform these operations on data we need a Database management system that allows us to store the data in such a way so that all these operations can be performed on the data efficiently.

Database systems are much better than traditional file processing systems which we have discussed in the separate article: DBMS vs File System.

Database Applications – DBMS

Applications where we use Database Management Systems are:

- **Telecom**: There is a database to keeps track of the information regarding calls made, network usage, customer details etc. Without the database systems it is hard to maintain that huge amount of data that keeps updating every millisecond.
- **Industry**: Where it is a manufacturing unit, warehouse or distribution centre, each one needs a database to keep the records of ins and outs. For example distribution centre should keep a track of the product units that supplied into the centre as well as the products that got delivered out from the distribution centre on each day; this is where DBMS comes into picture.
- **Banking System**: For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc. All this work has been done with the help of Database management systems.
- **Sales**: To store customer information, production information and invoice details.
- **Airlines**: To travel though airlines, we make early reservations, this reservation information along with flight schedule is stored in database.
- **Education sector**: Database systems are frequently used in schools and colleges to store and retrieve the data regarding student details, staff details, course details, exam details, payroll data, attendance details, fees details etc. There is a hell lot amount of inter-related data that needs to be stored and retrieved in an efficient manner.
- **Online shopping**: You must be aware of the online shopping websites such as Amazon, Flipkart etc. These sites store the product information, your addresses and preferences, credit details and provide you the relevant list of products based on your query. All this involves a Database management system.

Advantages of DBMS over file system

A file processing system and how Database management systems are better than file processing systems.

Drawbacks of File system

- **Data redundancy:** Data redundancy refers to the duplication of data, lets say we are managing the data of a college where a student is enrolled for two courses, the same student details in such case will be stored twice, which will take more storage than needed. Data redundancy often leads to higher storage costs and poor access time.
- **Data inconsistency:** Data redundancy leads to data inconsistency, lets take the same example that we have taken above, a student is enrolled for two courses and we have student address stored twice, now lets say student requests to change his address, if the address is changed at one place and not on all the records then this can lead to data inconsistency.

- **Data Isolation:** Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
- **Dependency on application programs:** Changing files would lead to change in application programs.
- **Atomicity issues:** Atomicity of a transaction refers to “All or nothing”, which means either all the operations in a transaction executes or none.
- **Data Security:** Data should be secured from unauthorised access, for example a student in a college should not be able to see the payroll details of the teachers, such kind of security constraints are difficult to apply in file processing systems.

Advantage of DBMS

There are several advantages of Database management system over file system. Few of them are as follows:

- **No redundant data**: Redundancy removed by data normalization. No data duplication saves storage and improves access time.
- **Data Consistency and Integrity**: As we discussed earlier the root cause of data inconsistency is data redundancy, since data normalization takes care of the data redundancy, data inconsistency also been taken care of as part of it
- **Data Security**: It is easier to apply access constraints in database systems so that only authorized user is able to access the data. Each user has a different set of access thus data is secured from the issues such as identity theft, data leaks and misuse of data.
- **Privacy**: Limited access means privacy of data.
- **Easy access to data** – Database systems manages data in such a way so that the data is easily accessible with fast response times.
- **Easy recovery**: Since database systems keeps the backup of data, it is easier to do a full recovery of data in case of a failure.
- **Flexible**: Database systems are more flexible than file processing systems.

Disadvantages of DBMS:

- DBMS implementation cost is high compared to the file system
- Complexity: Database systems are complex to understand
- Performance: Database systems are generic, making them suitable for various applications. However this feature affect their performance for some applications

DBMS Architecture

Database management systems architecture will help us understand the components of database system and the relation among them.

The architecture of DBMS depends on the computer system on which it runs. For example, in a client-server DBMS architecture, the database systems at server machine can run several requests made by client machine. We will understand this communication with the help of diagrams.

Types of DBMS Architecture

There are three types of DBMS architecture:

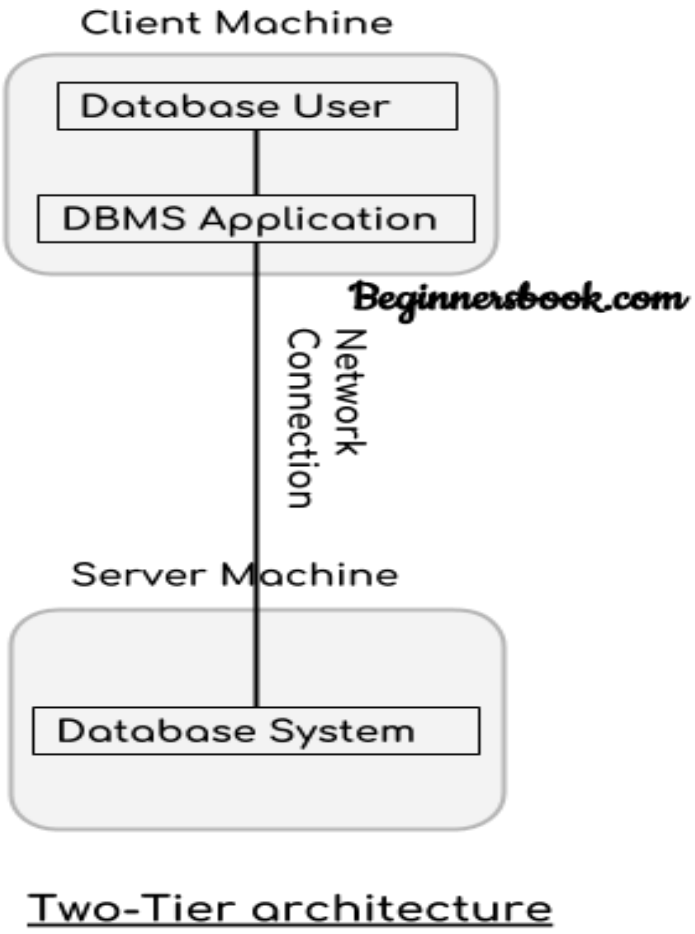
1. Single tier architecture
2. Two tier architecture
3. Three tier architecture

1. Single tier architecture

In this type of architecture, the database is readily available on the client machine, any request made by client doesn't require a network connection to perform the action on the database.

For example, lets say you want to fetch the records of employee from the database and the database is available on your computer system, so the request to fetch employee details will be done by your computer and the records will be fetched from the database by your computer as well. This type of system is generally referred as local database system.

2. Two tier architecture

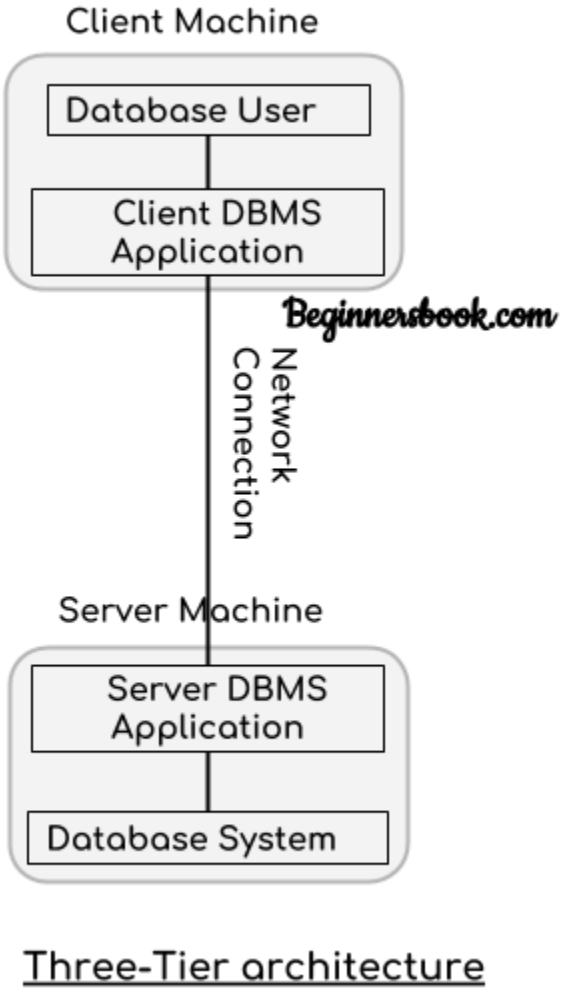


Two-Tier architecture

In two-tier architecture, the Database system is present at the server machine and the DBMS application is present at the client machine, these two machines are connected with each other through a reliable network as shown in the above diagram.

Whenever client machine makes a request to access the database present at server using a query language like sql, the server perform the request on the database and returns the result back to the client. The application connection interface such as JDBC, ODBC are used for the interaction between server and client.

3. Three tier architecture



Three-Tier architecture

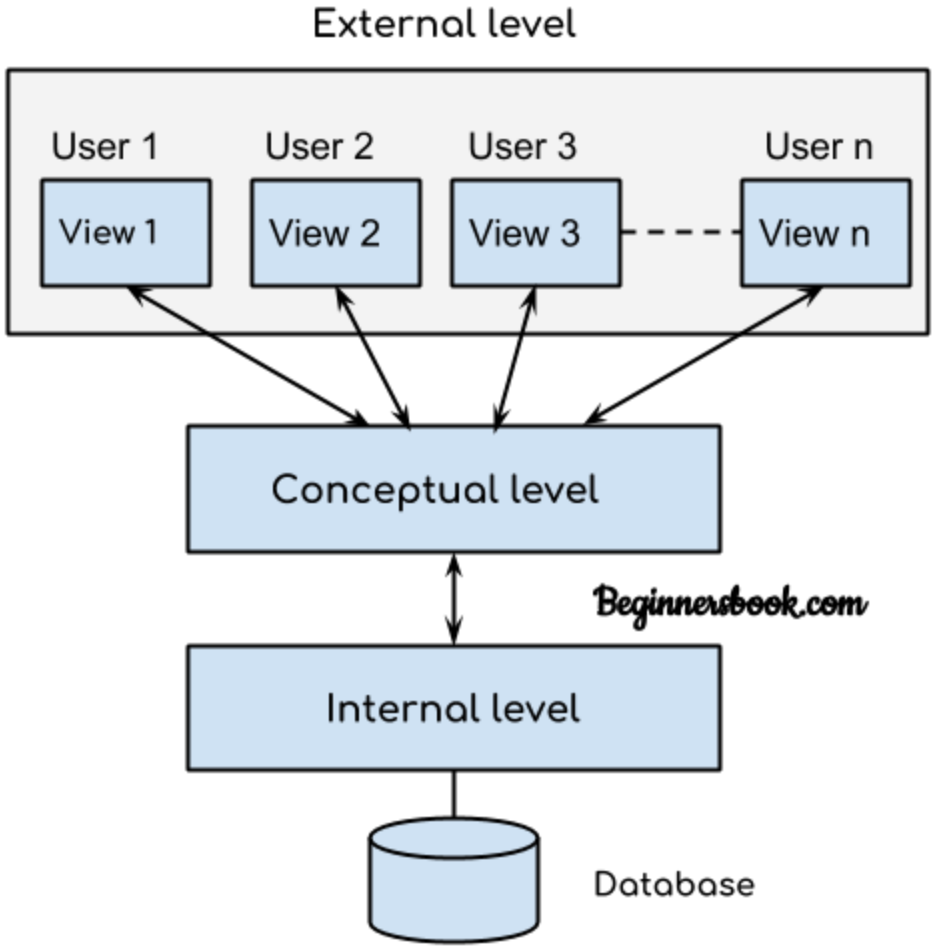
In three-tier architecture, another layer is present between the client machine and server machine. In this architecture, the client application doesn't communicate directly with the database systems present at the server machine, rather the client application

communicates with server application and the server application internally communicates with the database system present at the server.

DBMS – Three Level Architecture

The DBMS architecture – one-tier, two-tier and three-tier.

DBMS Three Level Architecture Diagram



This architecture has three levels:
1. External level
2. Conceptual level
3. Internal level

1. External level

It is also called **view level**. The reason this level is called “view” is because several users can view their desired data from this level which is internally fetched from database with the help of conceptual and internal level mapping.

the database schema details such as data structure, table definition etc. user is only concerned about data which is what returned back to the view level after it has been fetched from database (present at the internal level).

External level is the “**top level**” of the Three Level DBMS Architecture.

2. Conceptual level

It is also called **logical level**. The whole design of the database such as relationship among data, schema of data etc. are described in this level.

Database constraints and security are also implemented in this level of architecture. This level is maintained by DBA (database administrator).

3. Internal level

This level is also known as physical level. This level describes how the data is actually stored in the storage devices. This level is also responsible for allocating space to the data. This is the lowest level of the architecture.

View of Data in DBMS

Abstraction is one of the main features of database systems. Hiding irrelevant details from user and providing abstract view of data to users, helps in easy and efficient **user-database** interaction.

The top level of that architecture is “view level”. The view level provides the “**view of data**” to the users and hides the irrelevant details such as data relationship, database schema, constraints, security etc from the user.

To fully understand the view of data, you must have a basic knowledge of data abstraction and instance & schema.

1. Data abstraction
2. Instance and schema

DBMS Schema

Definition of schema: Design of a database is called the schema. Schema is of three types: Physical schema, logical schema and view schema.

The design of a database at physical level is called **physical schema**, how the data stored in blocks of storage is described at this level.

Design of database at logical level is called **logical schema**, programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation of data structure is hidden at this level (available at physical level).

Design of database at view level is called **view schema**. This generally describes end user interaction with database systems.

.

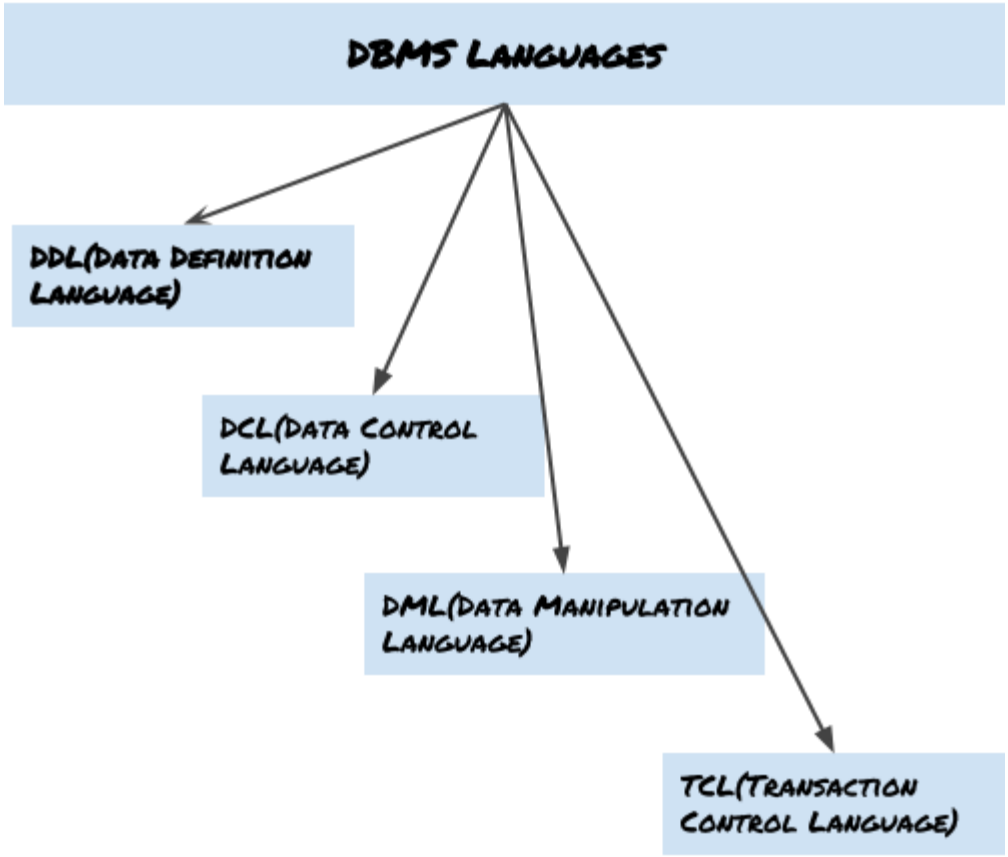
DBMS Instance

Definition of instance: The data stored in database at a particular moment of time is called instance of database. Database schema defines the variable declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database.

DBMS languages

Database languages are used to read, update and store data in a database. There are several such languages that can be used for this purpose; one of them is SQL (Structured Query Language).

Types of DBMS languages:



Data Definition Language (DDL)

DDL is used for specifying the database schema. It is used for creating tables, schema, indexes, constraints etc. in database. Let's see the operations that we can perform on database using DDL:

- To create the database instance – **CREATE**
- To alter the structure of database – **ALTER**
- To drop database instances – **DROP**
- To delete tables in a database instance – **TRUNCATE**
- To rename database instances – **RENAME**
- To drop objects from database such as tables – **DROP**
- To Comment – **Comment**

All of these commands either defines or update the database schema that's why they come under Data Definition language.

Data Manipulation Language (DML)

DML is used for accessing and manipulating data in a database. The following operations on database comes under DML:

- To read records from table(s) – SELECT
- To insert record(s) into the table(s) – **INSERT**
- Update the data in table(s) – UPDATE
- Delete all the records from the table – DELETE

Data Control language (DCL)

DCL is used for granting and revoking user access on a database –

- To grant access to user – GRANT
- To revoke access from user – REVOKE

In practical data definition language, data manipulation language and data control languages are not separate language, rather they are the parts of a single database language such as SQL.

Transaction Control Language(TCL)

The changes in the database that we made using DML commands are either performed or rollbacked using TCL.

- To persist the changes made by DML commands in database – COMMIT
- To rollback the changes made to the database – ROLLBACK

ENTITY RELATIONSHIP DIAGRAM – ER DIAGRAM IN DBMS

An **Entity–relationship model (ER model)** describes the structure of a database with the help of a diagram, which is known as **Entity Relationship Diagram (ER Diagram)**. An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

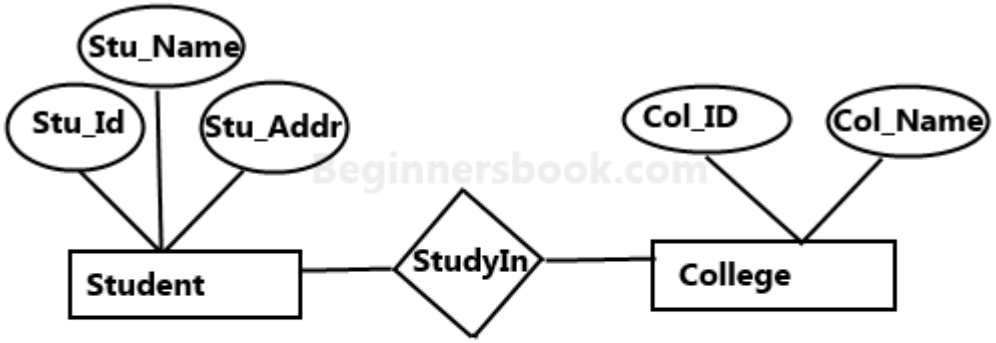
What is an Entity Relationship Diagram (ER Diagram):

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database.

A simple ER Diagram:

In the following diagram we have two entities Student and College and their relationship. The relationship between Student and College is many to one as a college can have many students however a student cannot study in multiple colleges at the same time.

Student entity has attributes such as Stu_Id, Stu_Name & Stu_Addr and College entity has attributes such as Col_ID & Col_Name.

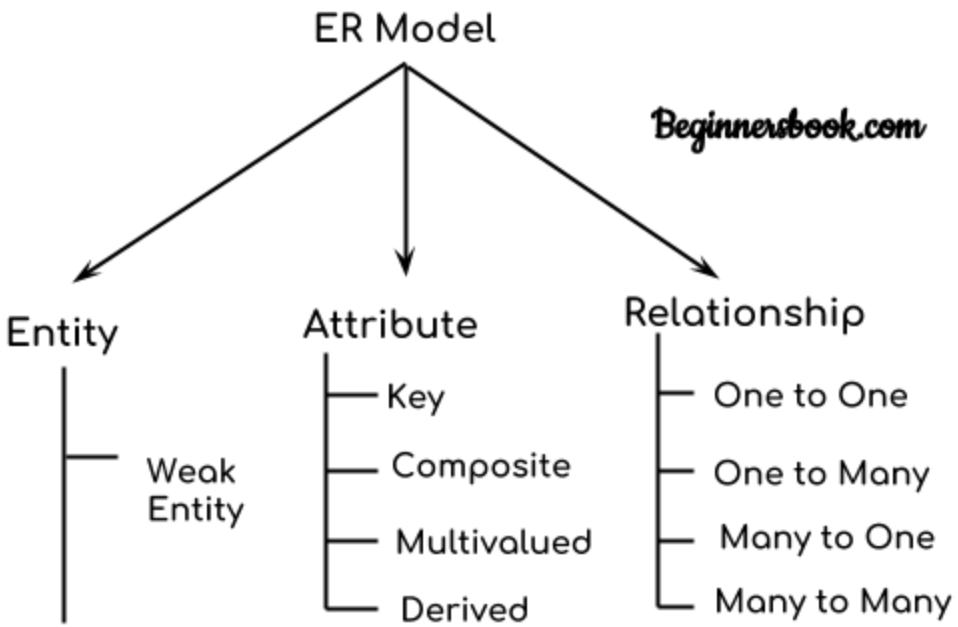


Sample E-R Diagram

Here are the geometric shapes and their meaning in an E-R Diagram.

Rectangle: Represents Entity sets.
Ellipses: Attributes
Diamonds: Relationship Set
Lines: They link attributes to Entity Sets and Entity sets to Relationship Set
Double Ellipses: Multivalued Attributes
Dashed Ellipses: Derived Attributes
Double Rectangles: Weak Entity Sets
Double Lines: Total participation of an entity in a relationship set

Components of a ER Diagram



Components of ER Diagram

As shown in the above diagram, an ER diagram has three main components:

1. Entity
2. Attribute
3. Relationship

1. Entity

An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.
For example: In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many students study in a single college. We will read more about relationships later, for now focus on entities.

Weak Entity:
An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity. The weak entity is represented by a double rectangle. For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.

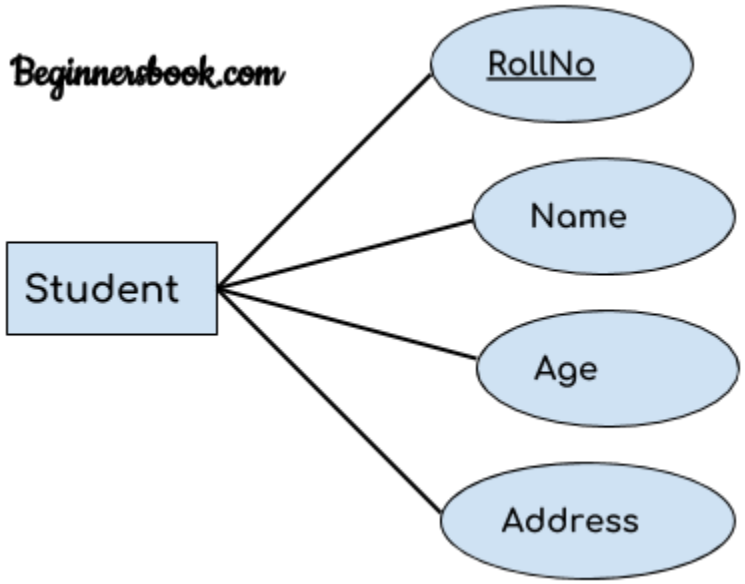
2. Attribute

An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram. There are four types of attributes:

1. Key attribute
2. Composite attribute
3. Multivalued attribute
4. Derived attribute

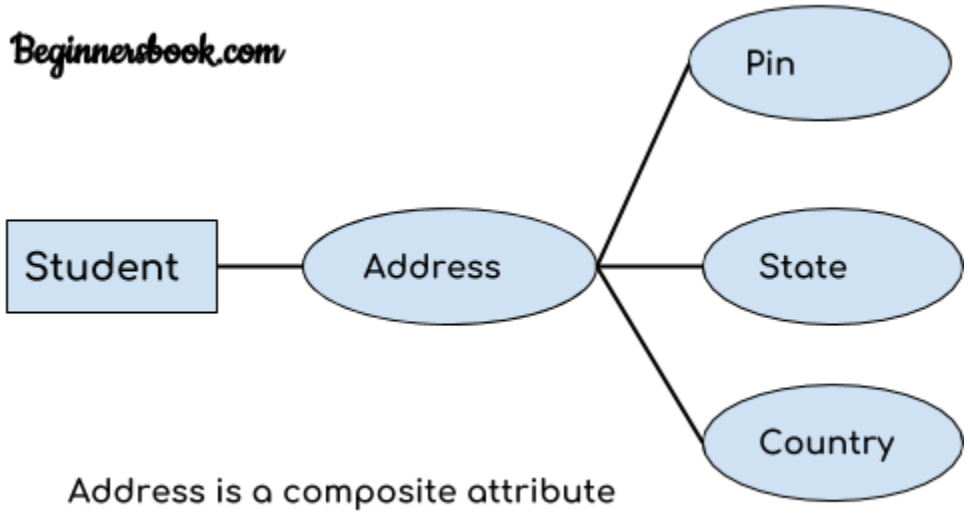
1. Key attribute:

A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students. Key attribute is represented by oval same as other attributes however the **text of key attribute is underlined**.



2. Composite attribute:

An attribute that is a combination of other attributes is known as composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.



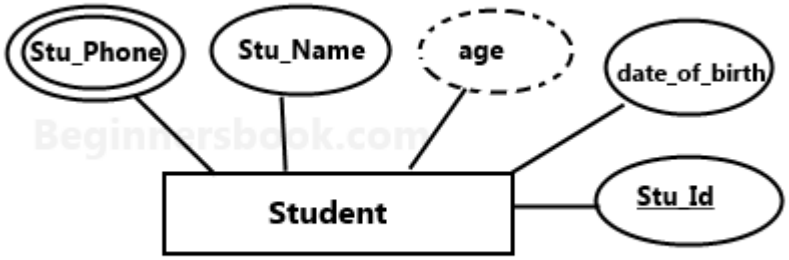
3. Multivalued attribute:

An attribute that can hold multiple values is known as multivalued attribute. It is represented with **double ovals** in an ER Diagram. For example – A person can have more than one phone numbers so the phone number attribute is multivalued.

4. Derived attribute:

A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by **dashed oval** in an ER Diagram. For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

E-R diagram with multivalued and derived attributes:



3. Relationship

A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. There are four types of relationships:

1. One to One
2. One to Many
3. Many to One
4. Many to Many

1. One to One Relationship

When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship. For example, a person has only one passport and a passport is given to one person.

Beginnersbook.com

2. One to Many Relationship

When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship. For example – a customer can place many orders but a order cannot be placed by many customers.

Beginnersbook.com

3. Many to One Relationship

When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.

Beginnersbook.com

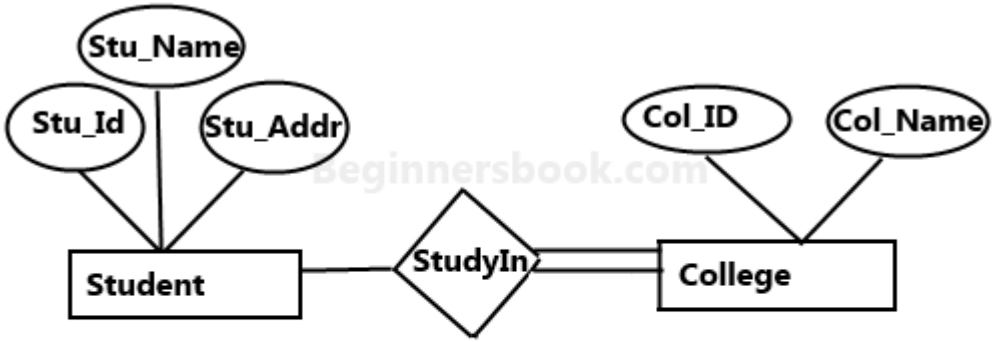
4. Many to Many Relationship

When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship. For example, a can be assigned to many projects and a project can be assigned to many students.

Beginnersbook.com

Total Participation of an Entity set

A Total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set. For example: In the below diagram each college must have at-least one associated Student.



E-R Digram with total participation of College entity set in StudyIn relationship Set - This indicates that each college must have atleast one associated Student.

KEYS IN DBMS

Key plays an important role in relational database; it is used for identifying unique rows from table. It also establishes relationship among tables.

Types of keys in DBMS

Primary Key – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.

Super Key – A super key is a set of one of more columns (attributes) to uniquely identify rows in a table.

Candidate Key – A super key with no redundant attribute is known as candidate key

Alternate Key – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.

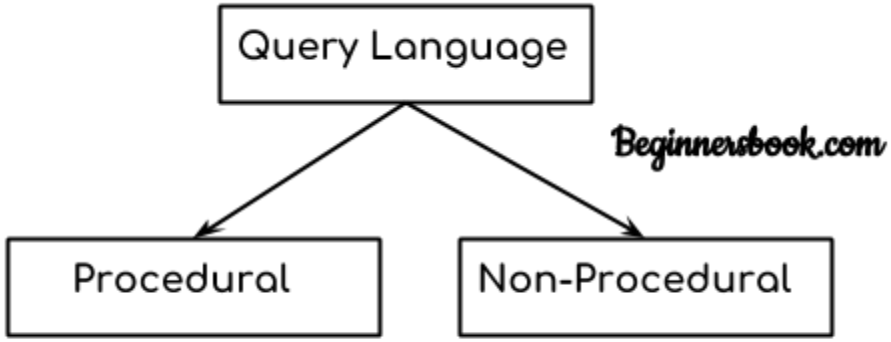
Composite Key – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.

Foreign Key – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

QUERY LANGUAGE

A Language which is used to store and retrieve data from database is known as query language. For example – **SQL**

There are two types of query language:
1.Procedural Query language
2.Non-procedural query language



1. Procedural Query language:

In procedural query language, user instructs the system to perform a series of operations to produce the desired results. Here users tells what data to be retrieved from database and how to retrieve it.

For example- to understand the procedural language, you are asking your younger brother to make a cup of tea, if you are just telling him to make a tea and not telling the process then it is a non-procedural language, however if you are telling the step by step process like switch on the stove, boil the water, add milk etc. then it is a procedural language.

2. Non-procedural query language:

In Non-procedural query language, user instructs the system to produce the desired result without telling the step by step process. Here users tells what data to be retrieved from database but doesn't tell how to retrieve it.

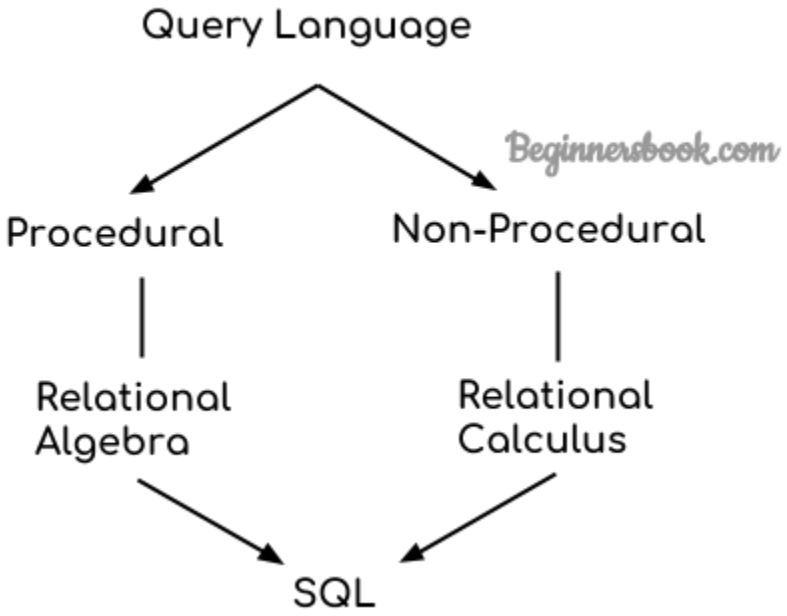
Relational Algebra:

Relational algebra is a conceptual procedural query language used on relational model.

Relational Calculus:

Relational calculus is a conceptual non-procedural query language used on relational model.

Note:
I have used word conceptual while describing relational algebra and relational calculus, because they are theoretical mathematical system or query language, they are not the practical implementation, SQL is a practical implementation of relational algebra and relational calculus.



Relational Algebra, Calculus, RDBMS & SQL:

Relational algebra and calculus are the theoretical concepts used on relational model.

RDBMS is a practical implementation of relational model.

SQL is a practical implementation of relational algebra and calculus.

In the next tutorials we will cover the relational algebra and calculus in detail.

DBMS Relational Algebra

In **Relational Algebra** the basics of relational algebra and calculus where we learned the need to use these theoretical mathematical systems.

What is Relational Algebra in DBMS?

Relational algebra is a **procedural** query language that works on relational model. The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, delete on the data. When I say that relational algebra is a procedural query language, it means that it tells what data to be retrieved and how to be retrieved.

On the other hand relational calculus is a non-procedural query language, which means it tells what data to be retrieved but doesn't tell how to retrieve it. We will discuss relational calculus in a separate tutorial.

Types of operations in relational algebra

We have divided these operations in two categories:
1. Basic Operations
2. Derived Operations

Basic/Fundamental Operations:

1. Select (σ)
2. Project (∏)
3. Union (∪)
4. Set Difference (-)
5. Cartesian product (X)
6. Rename (ρ)

Derived Operations:

1. Natural Join (⋈)
2. Left, Right, Full outer join (⟕, ⟖, ⟗)
3. Intersection (∩)
4. Division (÷)

Select Operator (σ)

Select Operator is denoted by sigma (σ) and it is used to find the tuples (or rows) in a relation (or table) which satisfy the given condition.

where clause in SQL, which is used for the same purpose.

Syntax of Select Operator (σ)

σ Condition/Predicate(Relation/Table name)

Select Operator (σ) Example

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

σ Customer_City="Agra" (CUSTOMER)

Output:

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra

Project Operator (∏)

Project operator is denoted by ∏ symbol and it is used to select desired columns (or attributes) from a table (or relation).

Project operator in relational algebra is similar to the Select statement in SQL.

Syntax of Project Operator (∏)

```
∏ column_name1, column_name2, ...., column_nameN(table_name)
```

Project Operator (∏) Example

In this example, we have a table CUSTOMER with three columns, we want to fetch only two columns of the table, which we can do with the help of Project Operator ∏.

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

```
∏ Customer_Name, Customer_City (CUSTOMER)
```

Output:

Customer_Name	Customer_City

```
Steve          Agra
Raghu          Agra
Chaitanya      Noida
Ajeet          Delhi
Carl           Delhi
```

Union Operator (∪)

Union operator is denoted by ∪ symbol and it is used to select all the rows (tuples) from two tables (relations).

we have two relations R1 and R2 both have same columns and we want to select all the tuples(rows) from these relations then we can apply the union operator on these relations.

Note: The rows (tuples) that are present in both the tables will only appear once in the union set. In short you can say that there are no duplicates present after the union operation.

Syntax of Union Operator (∪)

```
table_name1 ∪ table_name2
```

Union Operator (∪) Example

Table 1: COURSE

```
Course_Id  Student_Name   Student_Id
---------  ------------   ----------
C101       Aditya         S901
```

C104	Aditya	S901
C106	Steve	S911
C109	Paul	S921
C115	Lucy	S931

Table 2: STUDENT

Student_Id	Student_Name	Student_Age
S901	Aditya	19
S911	Steve	18
S921	Paul	19
S931	Lucy	17
S941	Carl	16
S951	Rick	18

Query:

∏ Student_Name (COURSE) ∪ ∏ Student_Name (STUDENT)

Output:

Student_Name

```
------------
Aditya
Carl
Paul
Lucy
Rick
Steve
```

Note: As you can see there are no duplicate names present in the output even though we had few common names in both the tables, also in the COURSE table we had the duplicate name itself.

Intersection Operator (∩)

Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).

Two relations R1 and R2 both have same columns and we want to select all those tuples(rows) that are present in both the relations, then in that case we can apply intersection operation on these two relations R1 ∩ R2.

Note: Only those rows that are present in both the tables will appear in the result set.

Syntax of Intersection Operator (∩)

```
table_name1 ∩ table_name2
```

Intersection Operator (∩) Example

Lets take the same example that we have taken above.
Table 1: COURSE

```
Course_Id  Student_Name   Student_Id
```

```
---------  ------------   ----------
C101       Aditya         S901
C104       Aditya         S901
C106       Steve          S911
C109       Paul           S921
C115       Lucy           S931
```

Table 2: STUDENT

```
Student_Id     Student_Name    Student_Age
------------   ----------      -----------
S901           Aditya          19
S911           Steve           18
S921           Paul            19
S931           Lucy            17
S941           Carl            16
S951           Rick            18
```

Query:

```
∏ Student_Name (COURSE) ∩ ∏ Student_Name (STUDENT)
```

Output:

```
Student_Name
------------
Aditya
Steve
Paul
Lucy
```

Set Difference (-)

Set Difference is denoted by – symbol. Lets say we have two relations R1 and R2 and we want to select all those tuples(rows) that are present in Relation R1 but **not** present in Relation R2, this can be done using Set difference R1 – R2.

Syntax of Set Difference (-)

```
table_name1 - table_name2
```

Set Difference (-) Example

Lets take the same tables COURSE and STUDENT that we have seen above.

Query:
Lets write a query to select those student names that are present in STUDENT table but not present in COURSE table.

```
∏ Student_Name (STUDENT) - ∏ Student_Name (COURSE)
```

Output:

```
Student_Name
------------
Carl
Rick
```

Cartesian product (X)

Cartesian Product is denoted by X symbol. Lets say we have two relations R1 and R2 then the cartesian product of these two relations (R1 X R2) would combine each tuple of first relation R1 with the each tuple of second relation R2. I know it sounds confusing but once we take an example of this, you will be able to understand this.

Syntax of Cartesian product (X)

```
R1 X R2
```

Cartesian product (X) Example

Table 1: R

```
Col_A    Col_B
-----    ------
AA       100
BB       200
CC       300
```

Table 2: S

```
Col_X     Col_Y
-----     -----
XX        99
YY        11
ZZ        101
```

Query:

Lets find the cartesian product of table R and S.

```
R X S
```

Output:

```
Col_A    Col_B     Col_X     Col_Y
-----    ------    ------    ------
AA       100       XX        99
AA       100       YY        11
AA       100       ZZ        101
```

BB	200	XX	99
BB	200	YY	11
BB	200	ZZ	101
CC	300	XX	99
CC	300	YY	11
CC	300	ZZ	101

Note: The number of rows in the output will always be the cross product of number of rows in each table. In our example table 1 has 3 rows and table 2 has 3 rows so the output has 3×3 = 9 rows.

Rename (ρ)

Rename (ρ) operation can be used to rename a relation or an attribute of a relation.
Rename (ρ) Syntax:
ρ(new_relation_name, old_relation_name)

Rename (ρ) Example

Lets say we have a table customer, we are fetching customer names and we are renaming the resulted relation to CUST_NAMES.

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

```
ρ(CUST_NAMES, ∏(Customer_Name)(CUSTOMER))
```

Output:

```
CUST_NAMES
----------
Steve
Raghu
Chaitanya
Ajeet
Carl
```

NORMALIZATION IN DBMS: 1NF, 2NF, 3NF AND BCNF IN DATABASE

Normalization is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly. Let's discuss about anomalies first then we will discuss normal forms with examples.

Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let's take an example to understand this.

Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

The above table is not normalized. We will see the problems that we face when a table is not normalized.

Update anomaly: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete anomaly: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

Normalization

Here are the most commonly used normal forms:

- First normal form(1NF)
- Second normal form(2NF)
- Third normal form(3NF)
- Boyce & Codd normal form (BCNF)

First normal form (1NF)

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Example: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

emp_id	emp_name	emp_address	emp_mobile

101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is **not in 1NF** as the rule says “each attribute of a table must have atomic (single) values”, the emp_mobile values for employees Jon & Lester violates that rule.

To make the table complies with 1NF we should have the data like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212

104	Lester	Bangalore	9990000123
104	Lester	Bangalore	8123450987

Second normal form (2NF)

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table.

An attribute that is not part of any candidate key is known as non-prime attribute.

Example: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}
Non prime attribute: teacher_age

The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says "**no** non-prime attribute is dependent on the proper subset of any candidate key of the table".

To make the table complies with 2NF we can break it in two tables like this:

teacher_details table:

teacher_id	teacher_age
111	38
222	38
333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics

222	Biology
333	Physics
333	Chemistry

Now the tables comply with Second normal form (2NF).

Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.

An attribute that is not part of any candidate key is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on
Candidate Keys: {emp_id}
Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

employee table:

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

employee_zip table:

emp_zip	emp_state	emp_city	emp_district

282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

Boyce Codd normal form (BCNF)

It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every functional dependency X->Y, X should be the super key of the table.

Example: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250

1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:
emp_nationality table:

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept table:

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping table:

emp_id	emp_dept
1001	Production and planning
1001	Stores
1002	design and technical support
1002	Purchasing department

Functional dependencies:

emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:
For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.

Functional dependency in DBMS

The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table.

For example: Suppose we have a student table with attributes: Stu_Id, Stu_Name, Stu_Age. Here Stu_Id attribute uniquely identifies the Stu_Name attribute of student table because if we know the student id we can tell the student name associated with it. This is known as functional dependency and can be written as Stu_Id->Stu_Name or in words we can say Stu_Name is functionally dependent on Stu_Id.

Formally:
If column A of a table uniquely identifies the column B of same table then it can represented as A->B (Attribute B is functionally dependent on attribute A)

Types of Functional Dependencies

- Trivial functional dependency
- non-trivial functional dependency
- Multivalued dependency
- Transitive dependency