UNIT- 1

1. WHAT IS SCHEMA, INSTANCE, METADATA, GENEARLIAZATION, , foreign key,SPECIALIZATION, data dictionary.

Ans – SCHEMA - A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them

INSTANCE - The situation where a data or information is stored in the database at a particular moment of time is called an instance. An instance is also called a current state or database state. The database schema that defines variables in tables which belong to a specific database, the records of these variables at a particular moment are called the instance of the database.

METADATA - **Metadata** is simply defined as data about data. It means it is a description and context of the data. It helps to organize, find and understand data. Let me explain to you by giving a real-world example of metadata:

Every time you take a photo with today’s cameras a bunch of metadata is gathered and saved with it. Such as

* File name,
* Size of the file,
* Date and time,
* Camera settings etc.

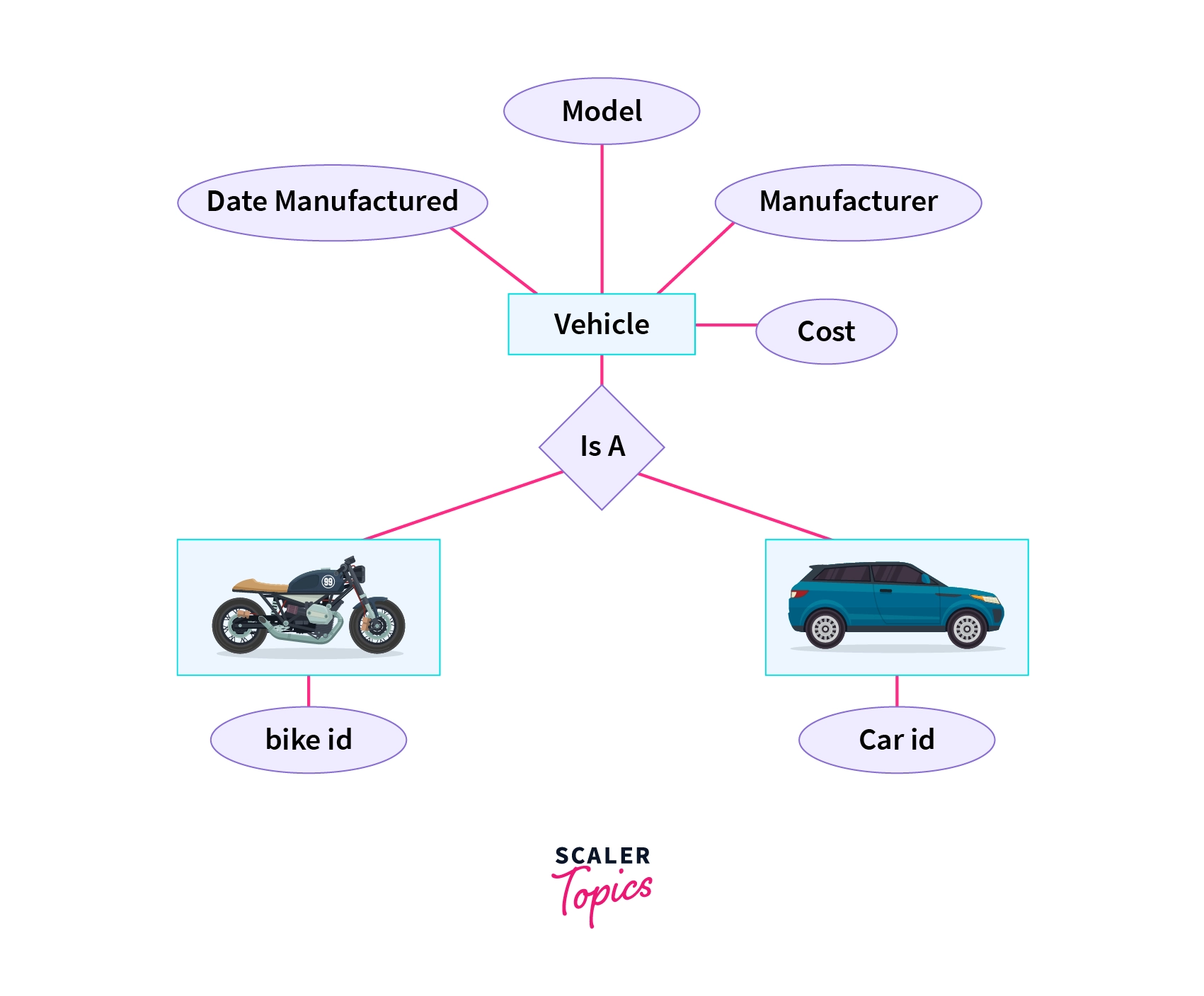
DATA DESCRIPTION - A data dictionary is a collection of descriptions of the data objects or items in a data model for the benefit of programmers and others who need to refer to them.

A data dictionary contains a list of all files in the database, the number of records in each file, and the names and types of each field. Most database management systems keep the data dictionary hidden from users to prevent them from accidentally destroying its contents.

FOREIGN KEY - A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables. It acts as a cross-reference between tables because it references the primary key of another table, thereby establishing a link between them.

GENERALIZATION - Generalization is a process in which a new entity is formed using the common attributes of two or more entities. Generalization simplifies the ER diagram by clubbing the common attributes together. Below are some characteristics of generalization:

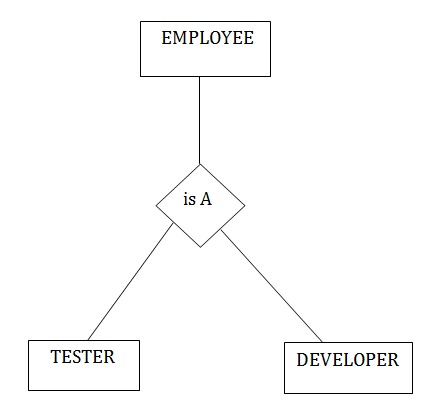
* Generalization follows the bottom-up approach.
* It generalizes or simplifies the entities.
* Higher-level entities can also be combined with lower-level entities.



SPECIALAIZATION - Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower level entities.

* Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.
* Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

**For example:** In an Employee management system, EMPLOYEE entity can be specialized as TESTER or DEVELOPER based on what role they play in the company.



1. WHAT IS DATA MODELS AND TYPES?

ANS - A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system. While the **Relational Model** is the most widely used database model, there are other models too:

* Hierarchical Model
* Network Model
* [Entity-relationship Model](https://www.studytonight.com/dbms/er-model-concepts.php)
* [Relational Model](https://www.studytonight.com/dbms/rdbms-concept.php)

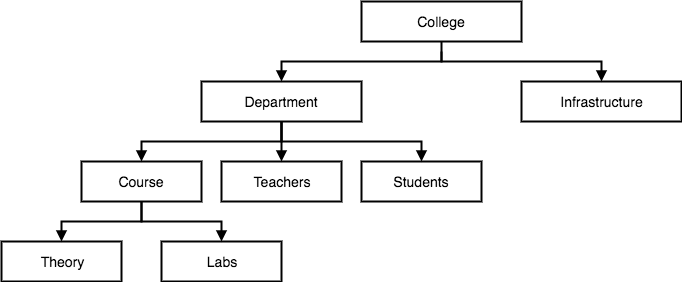
## Hierarchical Model

This database model organises data into a tree-like-structure, with a single root, to which all the other data is linked. The heirarchy starts from the **Root** data, and expands like a tree, adding child nodes to the parent nodes.

In this model, a child node will only have a single parent node.

This model efficiently describes many real-world relationships like index of a book, recipes etc.

In hierarchical model, data is organised into tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of-course many students.

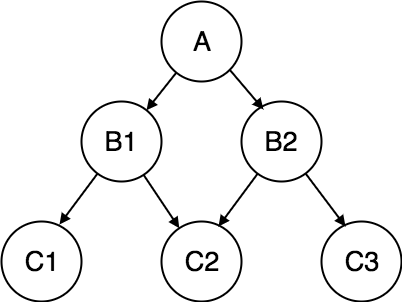


## Network Model

This is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node.

In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.

This was the most widely used database model, before Relational Model was introduced.



## Entity-relationship Model

In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes.

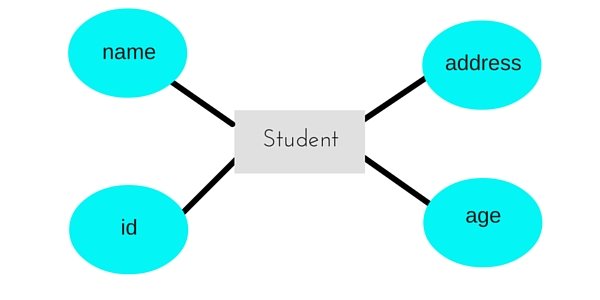
Different entities are related using relationships.

E-R Models are defined to represent the relationships into pictorial form to make it easier for different stakeholders to understand.

This model is good to design a database, which can then be turned into tables in relational model(explained below).

Let's take an example, If we have to design a School Database, then **Student** will be an **entity** with **attributes** name, age, address etc. As **Address** is generally complex, it can be another **entity** with **attributes** street name, pincode, city etc, and there will be a relationship between them.

Relationships can also be of different types. To learn about [E-R Diagrams](https://www.studytonight.com/dbms/er-diagram.php) in details, click on the link.



## Relational Model

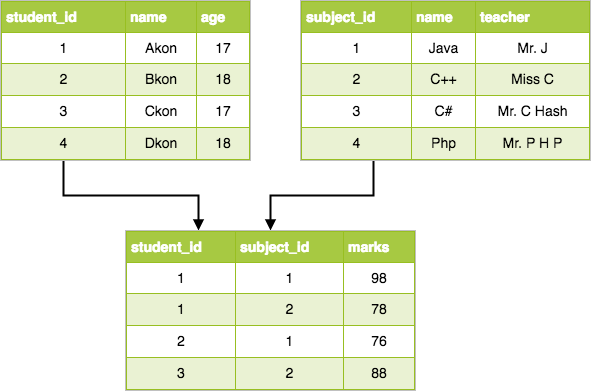
In this model, data is organised in two-dimensional **tables** and the relationship is maintained by storing a common field.

This model was introduced by E.F Codd in 1970, and since then it has been the most widely used database model, infact, we can say the only database model used around the world.

The basic structure of data in the relational model is tables. All the information related to a particular type is stored in rows of that table.

Hence, tables are also known as **relations** in relational model.

In the coming tutorials we will learn how to design tables, [normalize them](https://www.studytonight.com/dbms/database-normalization.php) to reduce data redundancy and how to use [Structured Query language](https://www.studytonight.com/dbms/introduction-to-sql.php) to access data from tables.



1. DBMS ARCHITECTURE?

ANS - Database Architecture is logically of two types:

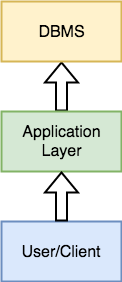
2-tier DBMS architecture

3-tier DBMS architecture

## 2-tier DBMS Architecture

2-tier DBMS architecture includes an **Application layer** between the user and the DBMS, which is responsible to communicate the user's request to the database management system and then send the response from the DBMS to the user.

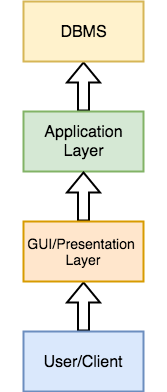
An application interface known as **ODBC**(Open Database Connectivity) provides an API that allow client side program to call the DBMS. Most DBMS vendors provide ODBC drivers for their DBMS.



Such an architecture provides the DBMS extra security as it is not exposed to the End User directly. Also, security can be improved by adding security and authentication checks in the Application layer too.

## 3-tier DBMS Architecture

3-tier DBMS architecture is the most commonly used architecture for web applications.



It is an extension of the 2-tier architecture. In the 2-tier architecture, we have an application layer which can be accessed programatically to perform various operations on the DBMS. The application generally understands the Database Access Language and processes end users requests to the DBMS.

In 3-tier architecture, an additional Presentation or GUI Layer is added, which provides a graphical user interface for the End user to interact with the DBMS.

For the end user, the GUI layer is the Database System, and the end user has no idea about the application layer and the DBMS system.

If you have used **MySQL**, then you must have seen **PHPMyAdmin**, it is the best example of a 3-tier DBMS architecture.

3. what is functions of dba? And define data independence?

# Ans - Functions and responsibilities of DBAs

**DBA:** person in the organization who controls the design and the use of the database refers as DBA.

### *1. Schema Definition:*

* The DBA definition the logical Schema of the database.A Schema refers to the overall logical structure of the database.
* According to this schema, database will be developed to store required data for an organization.

### *2. Storage Structure and Access Method Definition:*

* The DBA decides how the data is to be represented in the stored database.

### *3. Assisting Application Programmers:*

* The DBA provides assistance to application programmers to develop application programs.

### *4. Physical Organization Modification:*

* The DBA modifies the physical organization of the database to reflext the changing needs of the organization or to improve performance.

### *5. Approving Data Access:*

* The DBA determines which user needs access to which part of the database.
* According to this,various types of authorizations are granted to different users.

### *6. Monitoring Performance:*

* The DBA monitors performance of the system.The DBA ensures that better performance is maintained by making changes in physical or logical schema if required.

### *7. Backup and Recovery:*

* Database should not be lost or damaged.
* The DBA ensures this periodically backing up the database on magnetic tapes or remote servers.
* In case of failure, such as virus attack database is recovered from this backup.

Data independence is **the ability to modify the scheme without affecting the programs and the application to be rewritten**. Data is separated from the programs, so that the changes made to the data will not affect the program execution and the application.

1. Difference between file system and database management system?

Ans - • Data redundancy and inconsistency. Since different programmers create the

files and application programs over a long period, the various files are likely

to have different formats and the programs may be written in several programming languages. Moreover, the same information may be duplicated in

several places (files). For example, the address and telephone number of a particular customer may appear in a file that consists of savings-account records

and in a file that consists of checking-account records. This redundancy leads

to higher storage and access cost. In addition, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree. For

example, a changed customer address may be reflected in savings-account

records but not elsewhere in the system.

• Difficulty in accessing data. Suppose that one of the bank officers needs to

find out the names of all customers who live within a particular postal-code

area. The officer asks the data-processing department to generate such a list.

Because the designers of the original system did not anticipate this request,

there is no application program on hand to meet it. There is, however, an application program to generate the list of all customers. The bank officer hanow two choices: either obtain the list of all customers and extract the needed

information manually or ask a system programmer to write the necessary

application program. Both alternatives are obviously unsatisfactory. Suppose

that such a program is written, and that, several days later, the same officer

needs to trim that list to include only those customers who have an account

balance of $10,000 or more. As expected, a program to generate such a list does

not exist. Again, the officer has the preceding two options, neither of which is

satisfactory.

The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More

responsive data-retrieval systems are required for general use.

• 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.

• Integrity problems. The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount (say, $25). Developers enforce

these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult

to change the programs to enforce them. The problem is compounded when

constraints involve several data items from different files.

• Atomicity problems. A computer system, like any other mechanical or electrical device, is subject to failure. In many applications, it is crucial that, if a

failure occurs, the data be restored to the consistent state that existed prior to

the failure. Consider a program to transfer $50 from account A to account B.

If a system failure occurs during the execution of the program, it is possible

that the $50 was removed from account A but was not credited to account B,

resulting in an inconsistent database state. Clearly, it is essential to database

consistency that either both the credit and debit occur, or that neither occur.

That is, the funds transfer must be atomic—it must happen in its entirety or

not at all. It is difficult to ensure atomicity in a conventional file-processing

system.

• Concurrent-access anomalies. For the sake of overall performance of the system and faster response, many systems allow multiple users to update the

data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data. Consider bank account A, containing

$500. If two customers withdraw funds (say $50 and $100 respectively) from

account A at about the same time, the result of the concurrent executions may

leave the account in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce

that value by the amount being withdrawn, and write the result back. If the

two programs run concurrently, they may both read the value $500, and write

back $450 and $400, respectively.

• Security problems. Not every user of the database system should be able to

access all the data. For example, in a banking system, payroll personnel need

to see only that part of the database that has information about the various

bank employees. They do not need access to information about customer accounts. But, since application programs are added to the system in an ad hoc

manner, enforcing such security constraints is difficult.

UNIT – 5

1. Transactions ? Properties

or transaction processing?

Ans - A transaction is an action or series of actions that are being performed by a single user or application program, which reads or updates the contents of the database.

Transaction processing means dividing information processing up into individual, indivisible operations, called transactions, that complete or fail as a whole; a transaction can't remain in an intermediate, incomplete, state (so other processes can't access the transaction's data until either the transaction has completed or it has been "rolled back" after failure). Transaction processing is designed to maintain database integrity (the consistency of related data items) in a known, consistent state.

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

* **Atomicity** − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
* **Consistency** − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
* **Durability** − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
* **Isolation** − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

2. serializibiility?

Ans - Serializability

When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transactions are interleaved with some other transaction.

* **Schedule** − A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.
* **Serial Schedule** − It is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

3. what is distributed databases and explain its security mechanism>

ans – to be written

4. Conucrrent problem in transaction?

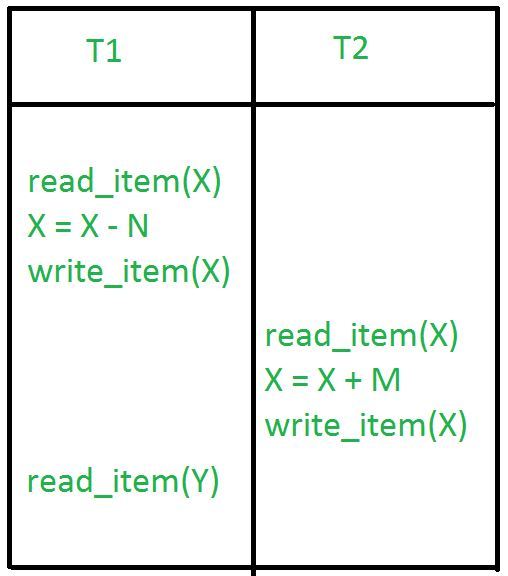
Ans - When [**multiple transactions**](https://www.geeksforgeeks.org/concurrency-control-in-dbms/) execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems. These problems are commonly referred to as concurrency problems in a database environment. The five concurrency problems that can occur in the database are:

* Temporary Update Problem
* Lost Update Problem
* Unrepeatable Read Problem

### Temporary Update Problem:

Temporary update or dirty read problem occurs when one transaction updates an item and fails. But the updated item is used by another transaction before the item is changed or reverted back to its last value.

#### Example:

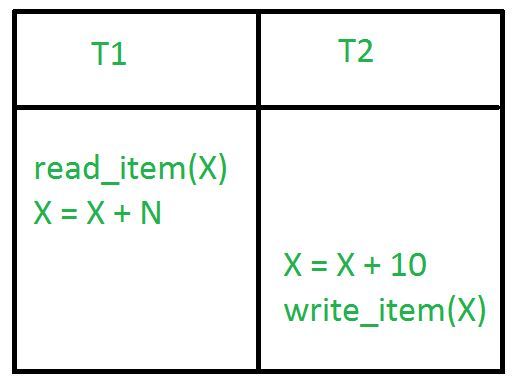


In the above example, if transaction 1 fails for some reason then X will revert back to its previous value. But transaction 2 has already read the incorrect value of X.

Lost Update Problem:

In the lost update problem, an update done to a data item by a transaction is lost as it is overwritten by the update done by another transaction.

#### Example:

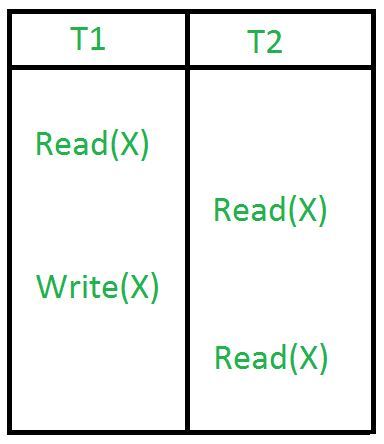


In the above example, transaction 1 changes the value of X but it gets overwritten by the update done by transaction 2 on X. Therefore, the update done by transaction 1 is lost.

### Unrepeatable Read Problem:

The unrepeatable problem occurs when two or more read operations of the same transaction read different values of the same variable.

#### Example:



In the above example, once transaction 2 reads the variable X, a write operation in transaction 1 changes the value of the variable X. Thus, when another read operation is performed by transaction 2, it reads the new value of X which was updated by transaction 1.

UNIT – 2 , 3

1. What is attributes, entity and its types?

Ans - An attribute is a property or characteristic of an entity. An entity may contain any number of attributes. One of the attributes is considered as the primary key. In an Entity-Relation model, attributes are represented in an elliptical shape.

There are five such types of attributes: Simple, Composite, Single-valued, Multi-valued, and Derived attribute.

### Simple attribute :

An attribute that cannot be further subdivided into components is a simple attribute.   
Example: The roll number of a student, the id number of an employee.

### Composite attribute :

An attribute that can be split into components is a composite attribute.

Example: The address can be further split into house number, street number, city, state, country, and pin code, the name can also be split into first name middle name, and last name.

### Single-valued attribute :

The attribute which takes up only a single value for each entity instance is a single-valued attribute.

Example: The age of a student.

### Multi-valued attribute :

The attribute which takes up more than a single value for each entity instance is a multi-valued attribute.

Example: Phone number of a student: Landline and mobile.

### Derived attribute :

An attribute that can be derived from other attributes is derived attributes.

Example: Total and average marks of a student.

Entity in DBMS can be a real-world object with an existence, For example, in a **College**database, the entities can be Professor, Students, Courses, etc.

Entities has attributes, which can be considered as properties describing it, for example, for Professor entity, the attributes are  **Professor\_Name, Professor\_Address, Professor\_Salary,** etc. The attribute value gets stored in the database.

## Example of Entity in DBMS

Let us see an example −

**<Professor>**

|  |  |  |  |
| --- | --- | --- | --- |
| **Professor\_ID** | **Professor\_Name** | **Professor\_City** | **Professor\_Salary** |
| P01 | Tom | Sydney | $7000 |
| P02 | David | Brisbane | $4500 |
| P03 | Mark | Perth | $5000 |

Here, **Professor\_Name, Professor \_Address and Professor \_Salary** are attributes.  
          **Professor\_ID** is the primary key

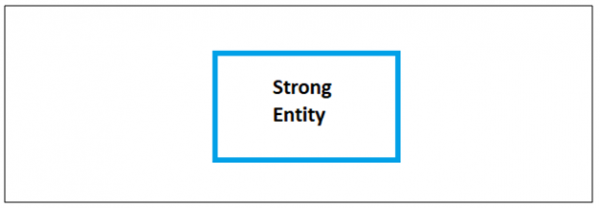
## Types of DBMS Entities

The following are the types of entities in DBMS −

## Strong Entity

The strong entity has a primary key. Weak entities are dependent on strong entity. Its existence is not dependent on any other entity.

Strong Entity is represented by a single rectangle −

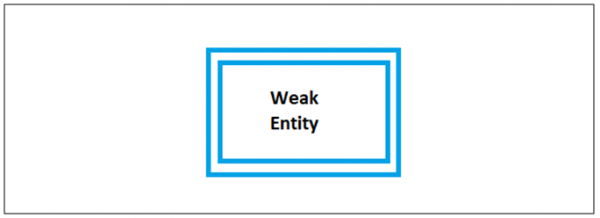


Continuing our previous example, **Professor**is a strong entity here, and the primary key is  **Professor\_ID.**

## Weak Entity

The weak entity in DBMS do not have a primary key and are dependent on the parent entity. It mainly depends on other entities.

Weak Entity is represented by double rectangle −



Continuing our previous example, **Professor**is a strong entity, and the primary key is  **Professor\_ID**. However, another entity is **Professor\_Dependents**, which is our Weak Entity.

**<Professor\_Dependents>**

|  |  |  |
| --- | --- | --- |
| **Name** | **DOB** | **Relation** |

This is a weak entity since its existence is dependent on another entity **Professor**, which we saw above. A Professor has Dependents.

3. explain all keys?

Ans - **1. Primary Key**

If you’re wondering what is primary key in DBMS,  primary key is a column of a table or a set of columns that helps to identify every record present in that table uniquely. There can be only one primary Key in a table. Also, the primary Key cannot have the same values repeating for any row. Every value of the primary key must be different with no repetitions. Amid many details, a primary key is the most significant one to understand what are keys and what is primary key in DBMS.

The PRIMARY KEY (PK) constraint put on a column or set of columns will not allow them to have any null values or any duplicates. One table can have only one primary key constraint. Any value in the primary key cannot be changed by any foreign keys (explained below) which refer to it.

### **2. Super Key**

Super Key is the set of all the keys which help to identify rows in a table uniquely. This means that all those columns of a table than capable of identifying the other columns of that table uniquely will all be considered super keys.

Super Key is the superset of a candidate key (explained below). The Primary Key of a table is picked from the super key set to be made the table’s identity attribute.

### **3. Candidate Key**

Candidate keys are those attributes that uniquely identify rows of a table. The Primary Key of a table is selected from one of the candidate keys. So, candidate keys have the same properties as the primary keys explained above. There can be more than one candidate keys in a table.

There can be more candidate keys than just one for any table, but they can never be empty. Every candidate key carries unique information and value. Besides these characteristics, a combination of attributes also works as a set of candidate keys.

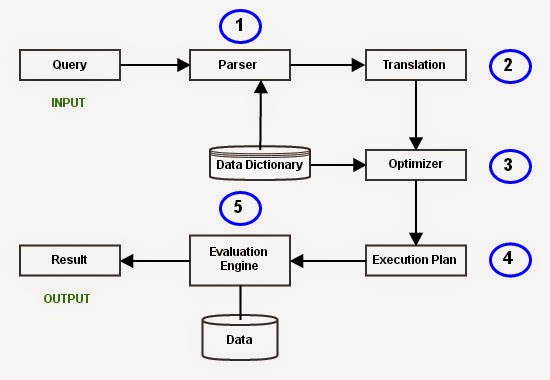
4. foreign key

FOREIGN KEY is a column that creates a relationship between two tables. The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity. It acts as a cross-reference between two tables as it references the primary key of another table.

4. query processing and its steps?

Ans - Query Processing

Query Processing would mean the entire process or activity which involves query translation into low level instructions, query optimization to save resources, cost estimation or evaluation of query, and extraction of data from the database.

Goal: To find an efficient Query Execution Plan for a given SQL query which would minimize the cost considerably, especially time.

**Step 1: Parsing**

In this step, the parser of the query processor module checks the syntax of the query, the user’s privileges to execute the query, the table names and attribute names, etc. The correct table names, attribute names and the privilege of the users can be taken from the system catalog (data dictionary).

**Step 2: Translation**

If we have written a valid query, then it is converted from high level language SQL to low level instruction in Relational Algebra.

**Step 3: Optimizer**

Optimizer uses the statistical data stored as part of data dictionary. The statistical data are information about the size of the table, the length of records, the indexes created on the table, etc. Optimizer also checks for the conditions and conditional attributes which are parts of the query.

**Step 4: Execution Plan**

A query can be expressed in many ways. The query processor module, at this stage, using the information collected in step 3 to find different relational algebra expressions that are equivalent and return the result of the one which we have written already.

**Step 5: Evaluation**

Though we got many execution plans constructed through statistical data, though they return same result (obvious), they differ in terms of Time consumption to execute the query, or the Space required executing the query. Hence, it is mandatory choose one plan which obviously consumes less cost.

At this stage, we choose one execution plan of the several we have developed. This Execution plan accesses data from the database to give the final result.

4. Relational algebra?

Ans – ma`am ne jo likhwaya h usme se pdhna h wo easy h

UNIT – 4

1. FUNCTIONAL DEPENEDCIES?

ANS - Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

1. X   →   Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

**For example:**

Assume we have an employee table with attributes: Emp\_Id, Emp\_Name, Emp\_Address.

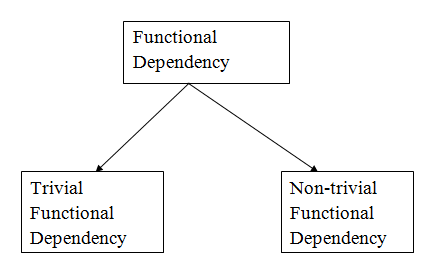
Here Emp\_Id attribute can uniquely identify the Emp\_Name attribute of employee table because if we know the Emp\_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. Emp\_Id → Emp\_Name

We can say that Emp\_Name is functionally dependent on Emp\_Id.

## Types of Functional dependency



### 1. Trivial functional dependency

* A → B has trivial functional dependency if B is a subset of A.
* The following dependencies are also trivial like: A → A, B → B

**Example:**

1. Consider a table with two columns Employee\_Id and Employee\_Name.
2. {Employee\_id, Employee\_Name}   →    Employee\_Id is a trivial functional dependency as
3. Employee\_Id is a subset of {Employee\_Id, Employee\_Name}.
4. Also, Employee\_Id → Employee\_Id and Employee\_Name   →    Employee\_Name are trivial dependencies too.

### 2. Non-trivial functional dependency

* A → B has a non-trivial functional dependency if B is not a subset of A.
* When A intersection B is NULL, then A → B is called as complete non-trivial.

**Example:**

1. ID   →    Name,
2. Name   →    DOB

2. what is normalization and its types?

Ans - What is Normalization?

* Normalization is the process of organizing the data in the database.
* Normalization is used to minimize 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.
* The normal form is used to reduce redundancy from the database table.

## First normal form (1NF)

A relation is said to be in **1NF (first normal form)**, if it doesn’t contain any multi-valued attribute. In other words you can say that a relation is in 1NF if each attribute contains only atomic(single) value only.

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**: Let’s say a company wants to store the names and contact details of its employees. It creates a table in the database that looks like this:

|  |  |  |  |
| --- | --- | --- | --- |
| Emp\_Id | Emp\_Name | Emp\_Address | Emp\_Mobile |
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212 , 9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Lester | Bangalore | 9990000123, 8123450987 |

Two employees (Jon & Lester) have two mobile numbers that caused the Emp\_Mobile field to have multiple values for these two employees.

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 need to create separate rows for the each mobile number in such a way so that none of the attributes contains multiple values.

|  |  |  |  |
| --- | --- | --- | --- |
| 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**](https://beginnersbook.com/2015/04/candidate-key-in-dbms/) of table.

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

**Example**: Let’s say a school wants to store the data of teachers and the subjects they teach. They create a table Teacher 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

This 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 disintegrate 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 are in 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**](https://beginnersbook.com/2015/04/transitive-dependency-in-dbms/) of non-prime attribute on any super key should be removed.

An attribute that is not part of any [**candidate key**](https://beginnersbook.com/2015/04/candidate-key-in-dbms/) 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**](https://beginnersbook.com/2015/04/super-key-in-dbms/) 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**: Let’s say 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. Further 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 disintegrate 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**](https://beginnersbook.com/2015/04/functional-dependency-in-dbms/) 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 table is now in BCNF as in both the functional dependencies left side part is a key.