

Contents

Unit 1: Introduction L.H.4.....	4
1.1 Definition of Database and database system	4
1.2 Characteristic of database approach	4
1.3 Database system versus Traditional file processing system.....	5
1.4 Advantage of Database Management System	6
1.4 Limitation of Database Management System	7
Unit II: Database System Concepts and Architecture	9
2.1 Database Models.....	9
2.2 DBMS Architecture	13
2.3 Database Language	17
2.4 Database users and Database administrator.....	18
2.5 E-R Model: Entities, Attributes, Relationships, Keys, Cardinalities, Participation constraints, E-R Diagram.....	20
Super Key	25
Candidate Key.....	25
Primary Key	26
Composite Key.....	26
ER-Diagram	28
Data Dictionary	35
Unit 3: Relation Model	36
3.1 Properties of relation.....	36
3.2 Schemas.....	37
3.3 Relation Algebra	38
Unit 4: SQL.....	41

4.1 Introduction to SQL, DDL and DML	41
4.2 Basic Structure of SQL Statement and SQL Queries	42
4.4 Joined relation, Subqueries	55
4.5 Sets Operation.....	56
4.6 Views	58
Unit 5 Integrity Constraints	60
1.Domain constraints	61
2.Entity integrity constraints.....	61
3.Referential Integrity Constraints.....	62
4. Key constraints	62
Unit VI: Relational Database Design	64
6.1 Pitfall of Relational Model.....	64
Data Redundancy	64
Anomalies.....	64
6.2 Functional Dependency	65
6.3 Normalization, Its need and objective	65
6.4 1NF,2NF,3NF,BCNF and 4NF	66
Unit VII: Database Security	67
7.1 Access Control.....	67
Discretionary and Mandatory Access Control.....	67
7.2 Authorization and Authentication.....	67
7.3 Data Encryption and Decryption.....	69
Unit VIII: Transaction Management, Recovery, and Query Processing	70
8.1 Introduction to Transaction	70
ACID Properties.....	70

8.2 Introduction to concurrency control.....	71
8.3 Reason for transaction failure	71
System Recovery and Media Recovery.....	72
8.4 Introduction to query processing.....	73
Step in Query Processing	74

Unit 1: Introduction L.H.4

1.1 Definition of Database and database system

Database is a collection of related data and data is a collection of facts and figures that can be processed to produce information.

Mostly data represents recordable facts. Data aids in producing information, which is based on facts. For example, if we have data about marks obtained by all students, we can then conclude about toppers and average marks.

A database management system stores data in such a way that it becomes easier to retrieve, manipulate, and produce information

1.2 Characteristic of database approach

Traditionally, data was organized in file formats. DBMS was a new concept then, and all the research was done to make it overcome the deficiencies in traditional style of data management. A modern DBMS has the following characteristics:

- **Real-world entity:** A modern DBMS is more realistic and uses real-world entities to design its architecture. It uses the behavior and attributes too. For example, a school database may use students as an entity and their age as an attribute.
- **Relation-based tables:** DBMS allows entities and relations among them to form tables. A user can understand the architecture of a database just by looking at the table names.
- **Less redundancy:** DBMS follows the rules of normalization, which splits a relation when any of its attributes is having redundancy in values. Normalization is a mathematically rich and scientific process that reduces data redundancy.
- **Query Language:** DBMS is equipped with query language, which makes it more efficient to retrieve and manipulate data. A user can apply as many and as different

filtering options as required to retrieve a set of data. Traditionally it was not possible where file-processing system was used.

- **Multuser and Concurrent Access:** DBMS supports multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when users attempt to handle the same data item, but users are always unaware of them.
- **Multiple views:** DBMS offers multiple views for different users. A user who is in the Sales department will have a different view of database than a person working in the Production department. This feature enables the users to have a concentrate view of the database according to their requirements.
- **Security:** Features like multiple views offer security to some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into the database and retrieving the same at a later stage. DBMS offers many different levels of security features, which enables multiple users to have different views with different features. For example, a user in the Sales department cannot see the data that belongs to the Purchase department. Additionally, it can also be managed how much data of the Sales department should be displayed to the user. Since a DBMS is not saved on the disk as traditional file systems, it is very hard for miscreants to break the code

1.3 Database system versus Traditional file processing system

1. A file processing system is a collection of programs that store and manage files in computer hard-disk. On the other hand, a database management system is collection of programs that enables to create and maintain a database.
2. File processing system has more data redundancy, less data redundancy in DBMS.
3. File processing system provides less flexibility in accessing data, whereas DBMS has more flexibility in accessing data.

4. File processing system does not provide data consistency, whereas DBMS provides data consistency through normalization.
5. File processing system is less complex, whereas DBMS is more complex.

1.4 Advantage of Database Management System

A Database Management System (DBMS) is defined as the software system that allows users to define, create, maintain and control access to the database. DBMS makes it possible for end users to create, read, update and delete data in database. It is a layer between programs and data.

Compared to the File Based Data Management System, Database Management System has many advantages. Some of these advantages are given below:

Reducing Data Redundancy

The file based data management systems contained multiple files that were stored in many different locations in a system or even across multiple systems. Because of this, there were sometimes multiple copies of the same file which lead to data redundancy.

This is prevented in a database as there is a single database and any change in it is reflected immediately. Because of this, there is no chance of encountering duplicate data.

Sharing of Data

In a database, the users of the database can share the data among themselves. There are various levels of authorization to access the data, and consequently the data can only be shared based on the correct authorization protocols being followed.

Many remote users can also access the database simultaneously and share the data between themselves.

Data Integrity

Data integrity means that the data is accurate and consistent in the database. Data Integrity is very important as there are multiple databases in a DBMS. All of these

databases contain data that is visible to multiple users. So it is necessary to ensure that the data is correct and consistent in all the databases and for all the users.

Data Security

Data Security is vital concept in a database. Only authorized users should be allowed to access the database and their identity should be authenticated using a username and password. Unauthorized users should not be allowed to access the database under any circumstances as it violates the integrity constraints.

Privacy

The privacy rule in a database means only the authorized users can access a database according to its privacy constraints. There are levels of database access and a user can only view the data he is allowed to. For example - In social networking sites, access constraints are different for different accounts a user may want to access.

Backup and Recovery

Database Management System automatically takes care of backup and recovery. The users don't need to backup data periodically because this is taken care of by the DBMS. Moreover, it also restores the database after a crash or system failure to its previous condition.

1.4 Limitation of Database Management System

Database Management System is quite useful compared to the file based management system. However, it does have some disadvantages. Some of those are as follows:

More Costly

Creating and managing a database is quite costly. High cost software and hardware is required for the database. Also highly trained staff is required to handle the database and it also needs continuous maintenance. All of these ends up making a database quite a costly venture.

High Complexity

A Database Management System is quite complex as it involves creating, modifying and editing a database. So, the people who handle a database or work with it need to be quite skilled or valuable data can be lost.

Database handling staff required

As discussed in the previous point, database and DBMS are quite complex. Hence, skilled personnel are required to handle the database so that it works in optimum condition. This is a costly venture as these professionals need to be very well paid.

Database Failure

All the relevant data for any company is stored in a database. So it is imperative that the database works in optimal condition and there are no failures. A database failure can be catastrophic and can lead to loss or corruption of very important data.

High Hardware Cost

A database contains vast amount of data. So a large disk storage is required to store all this data. Sometimes extra storage may even be needed. All this increases hardware costs by a lot and makes a database quite expensive.

Upgradation Costs

Often new functionalities are added to the database. This leads to database upgradations. All of these upgradations cost a lot of money. Moreover, it is also quite expensive to train the database managers and users to handle these new upgradations.

Unit II: Database System Concepts and Architecture

2.1 Database Models

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
- Relational Model

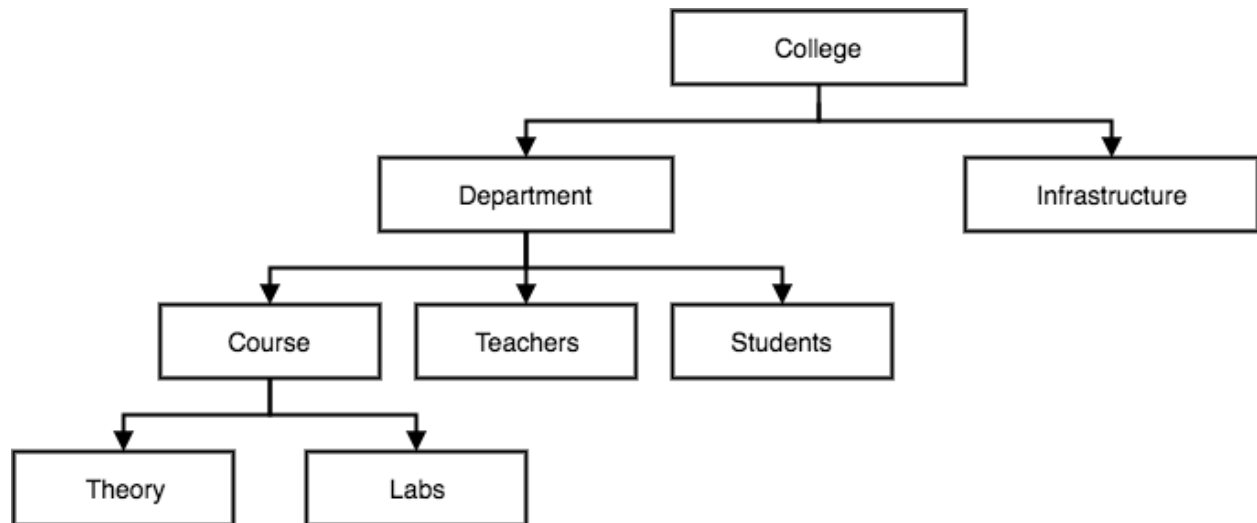
Hierarchical Model

This database model organizes data into a tree-like-structure, with a single root, to which all the other data is linked. The hierarchy 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 organized 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 off-course many students.

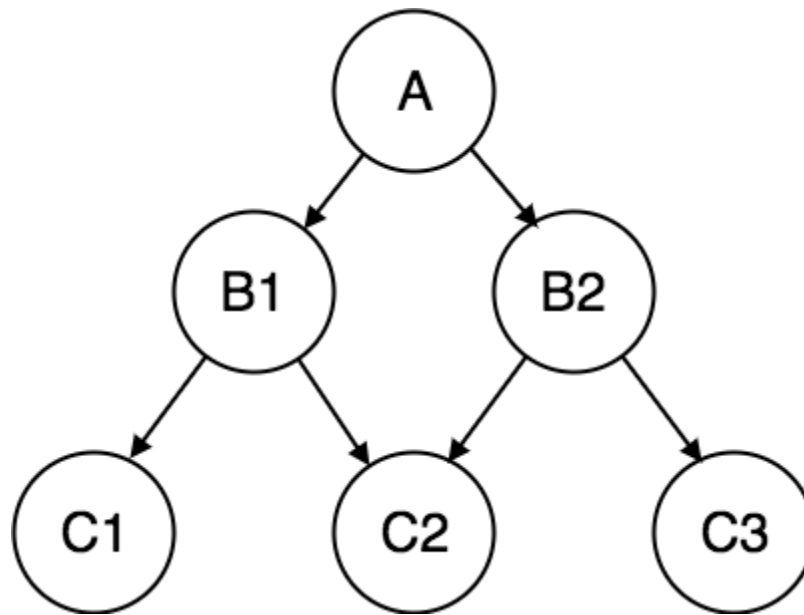


Network Model

This is an extension of the Hierarchical model. In this model data is organized 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

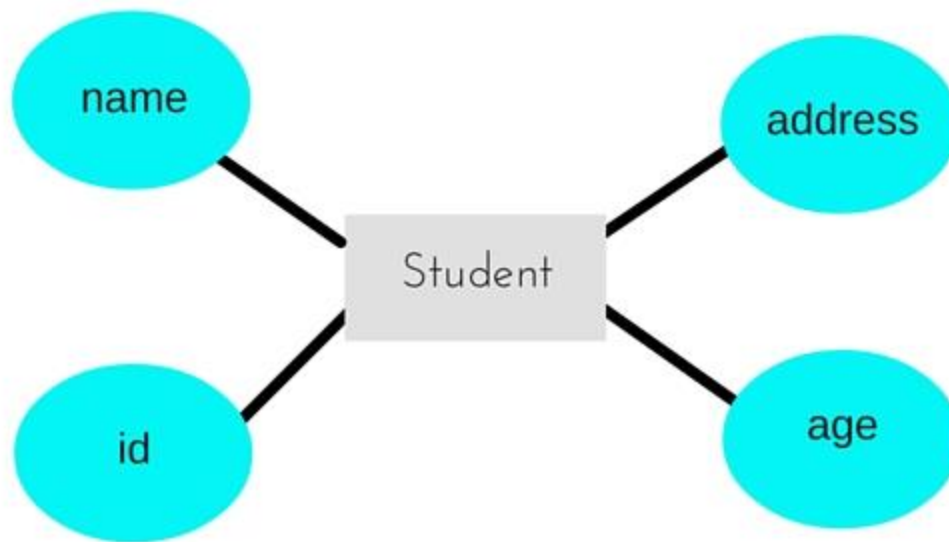
In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes.

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, pin code, city etc, and there will be a relationship between them.



Relational Model

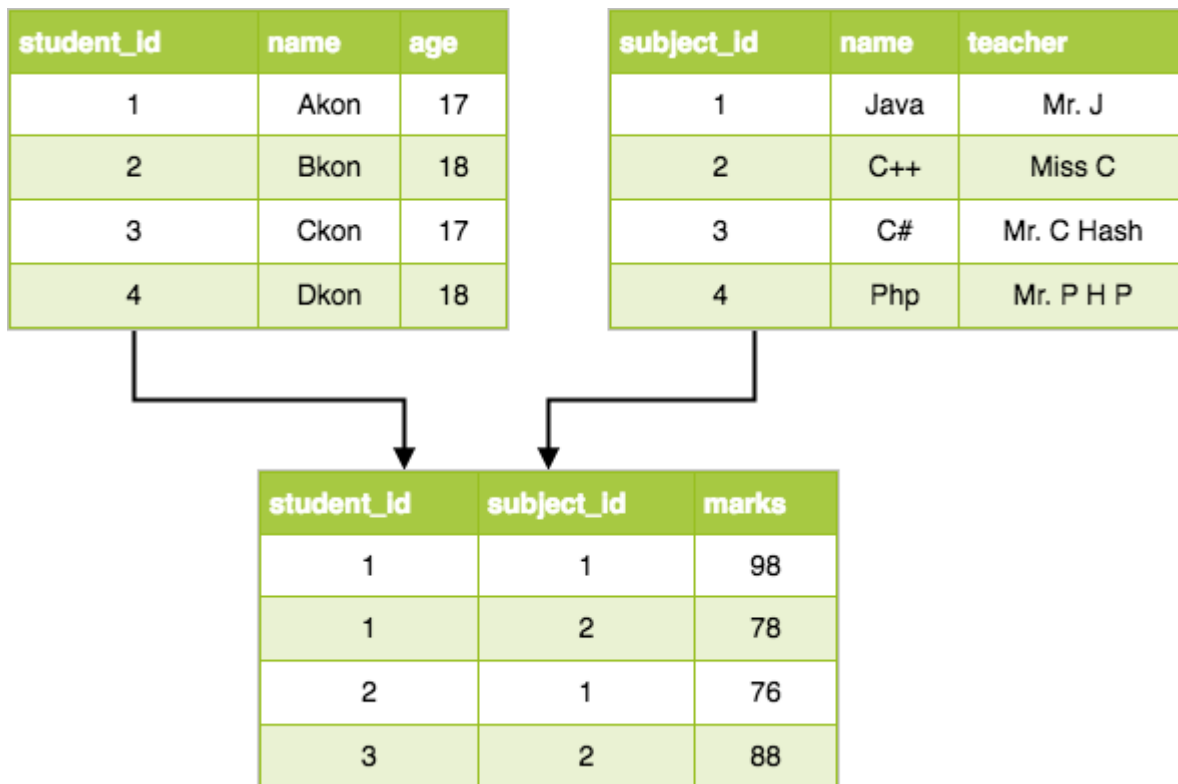
In this model, data is organized in two-dimensional **tables** and the relationship is maintained by storing a common field.

This model was introduced by E.F Codd in 1970, and since then it has been the most widely used database model, in fact, 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 to reduce data redundancy and how to use Structured Query language to access data from tables.



2.2 DBMS Architecture

A Database Management system is not always directly available for users and applications to access and store data in it. A Database Management system can be **centralized** (all the data stored at one location), **decentralized** (multiple copies of database at different locations) or **hierarchical**, depending upon its architecture.

1-tier DBMS architecture also exist; this is when the database is directly available to the user for using it to store data. Generally, such a setup is used for local application development, where programmers communicate directly with the database for quick response.

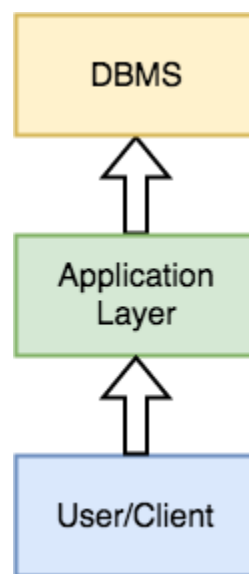
Database Architecture is logically of two types:

1. 2-tier DBMS architecture
2. 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.

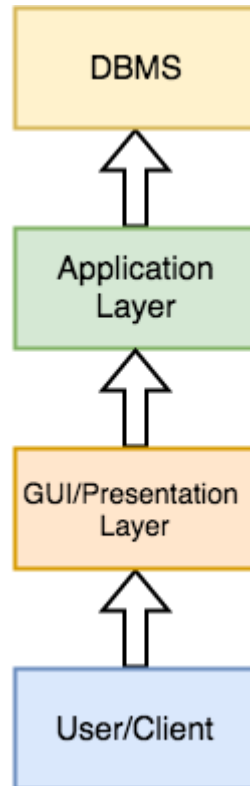
An application interface known as **ODBC** (Open Database Connectivity) provides an API that allow client side program to call the DBMS. Most DBMS vendors provide ODBC drivers for their DBMS.



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 programmatically 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.

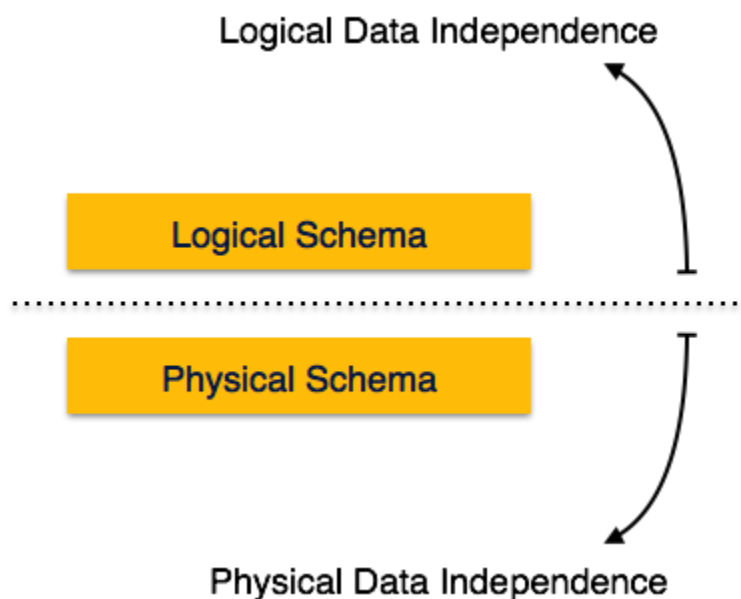
For the end user, the GUI layer is the Database System, and the end user has no idea about the application layer and the DBMS system.

2.2 Data independence

If a database system is not multi-layered, then it becomes difficult to make any changes in the database system. Database systems are designed in multi-layers as we learnt earlier.

Data Independence

A database system normally contains a lot of data in addition to users' data. For example, it stores data about data, known as metadata, to locate and retrieve data easily. It is rather difficult to modify or update a set of metadata once it is stored in the database. But as a DBMS expands, it needs to change over time to satisfy the requirements of the users. If the entire data is dependent, it would become a tedious and highly complex job.



Metadata itself follows a layered architecture, so that when we change data at one layer, it does not affect the data at another level. This data is independent but mapped to each other.

Logical Data Independence

Logical data is data about database, that is, it stores information about how data is managed inside. For example, a table (relation) stored in the database and all its constraints, applied on that relation.

Logical data independence is a kind of mechanism, which liberalizes itself from actual data stored on the disk. If we do some changes on table format, it should not change the data residing on the disk

Physical Data Independence

All the schemas are logical, and the actual data is stored in bit format on the disk. Physical data independence is the power to change the physical data without impacting the schema or logical data.

For example, in case we want to change or upgrade the storage system itself – suppose we want to replace hard-disks with SSD – it should not have any impact on the logical data or schemas.

2.3 Database Language

A DBMS must provide appropriate languages and interfaces for each category of users to express database queries and updates. Database Languages are used to create and maintain database on computer. There are large numbers of database languages like Oracle, MySQL, MS Access, dBase, FoxPro etc. SQL statements commonly used in Oracle and MS Access can be categorized as data definition language (DDL), data control language (DCL) and data manipulation language (DML).

2.4 Database users and Database administrator

Database Users

Database users are the one who really use and take the benefits of database. There will be different types of users depending on their need and way of accessing the database.

1. **Application Programmers** - They are the developers who interact with the database by means of DML queries. These DML queries are written in the application programs like C, C++, JAVA, Pascal etc. These queries are converted into object code to communicate with the database. For example, writing a C program to generate the report of employees who are working in particular department will involve a query to fetch the data from database. It will include a embedded SQL query in the C Program.
2. **Sophisticated Users** - They are database developers, who write SQL queries to select/insert/delete/update data. They do not use any application or programs to request the database. They directly interact with the database by means of query language like SQL. These users will be scientists, engineers, analysts who thoroughly study SQL and DBMS to apply the concepts in their requirement. In short, we can say this category includes designers and developers of DBMS and SQL.
3. **Specialized Users** - These are also sophisticated users, but they write special database application programs. They are the developers who develop the complex programs to the requirement.
4. **Stand-alone Users** - These users will have stand –alone database for their personal use. These kinds of database will have readymade database packages which will have menus and graphical interfaces.
5. **Native Users** - these are the users who use the existing application to interact with the database. For example, online library system, ticket booking systems, ATMs etc which has existing application and users use them to interact with the database to fulfill their requests.

Database Administrator

The life cycle of database starts from designing, implementing to administration of it. A database for any kind of requirement needs to be designed perfectly so that it should work without any issues. Once all the design is complete, it needs to be installed. Once this step is complete, users start using the database. The database grows as the data grows in the database. When the database becomes huge, its performance comes down. Also accessing the data from the database becomes challenge. There will be unused memory in database, making the memory inevitably huge. These administration and maintenance of database is taken care by database Administrator – DBA. A DBA has many responsibilities. A good performing database is in the hands of DBA.

1. **Installing and upgrading the DBMS Servers:** - DBA is responsible for installing a new DBMS server for the new projects. He is also responsible for upgrading these servers as there are new versions comes in the market or requirement. If there is any failure in upgradation of the existing servers, he should be able revert the new changes back to the older version, thus maintaining the DBMS working. He is also responsible for updating the service packs/ hot fixes/ patches to the DBMS servers.
2. **Design and implementation:** - Designing the database and implementing is also DBA's responsibility. He should be able to decide proper memory management, file organizations, error handling, log maintenance etc for the database.
3. **Performance tuning:** - Since database is huge and it will have lots of tables, data, constraints and indices, there will be variations in the performance from time to time. Also, because of some designing issues or data growth, the database will not work as expected. It is responsibility of the DBA to tune the database performance. He is responsible to make sure all the queries and programs works in fraction of seconds.
4. **Migrate database servers:** - Sometimes, users using oracle would like to shift to SQL server. It is the responsibility of DBA to make sure that migration happens without any failure, and there is no data loss.
5. **Backup and Recovery:** - Proper backup and recovery programs needs to be developed by DBA and has to be maintained him. This is one of the main

responsibilities of DBA. Data/objects should be backed up regularly so that if there is any crash, it should be recovered without much effort and data loss.

6. **Security:** - DBA is responsible for creating various database users and roles, and giving them different levels of access rights.
7. **Documentation:** - DBA should be properly documenting all his activities so that if he quits or any new DBA comes in, he should be able to understand the database without any effort. He should basically maintain all his installation, backup, recovery, security methods. He should keep various reports about database performance.

2.5 E-R Model: Entities, Attributes, Relationships, Keys, Cardinalities, Participation constraints, E-R Diagram

ER-Model

The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

Entity

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise, a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Attributes

Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

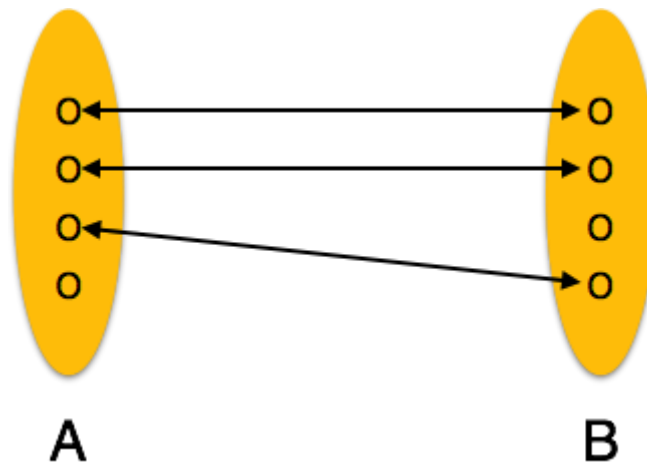
- ✓ **Simple attribute** – Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- ✓ **Composite attribute** – Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.
- ✓ **Derived attribute** – Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.
- ✓ **Single-value attribute** – Single-value attributes contain single value. For example – Social_Security_Number.
- ✓ **Multi-value attribute** – Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

Relationship

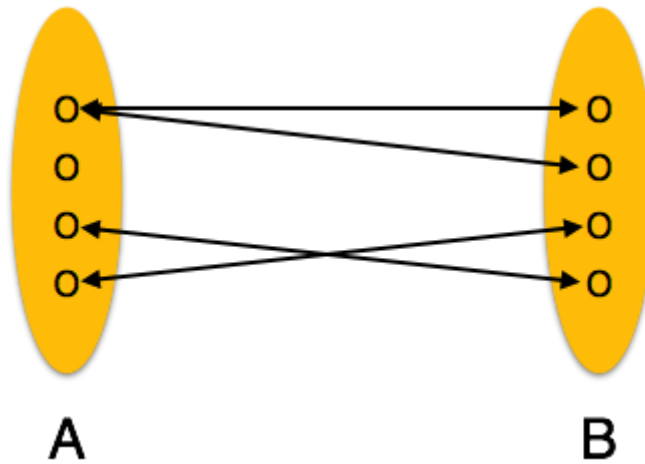
The association among entities is called a relationship. For example, an employee **works_at** a department, a student **enrolls** in a course. Here, Works_at and Enrolls are called relationships.

Types of Relationship

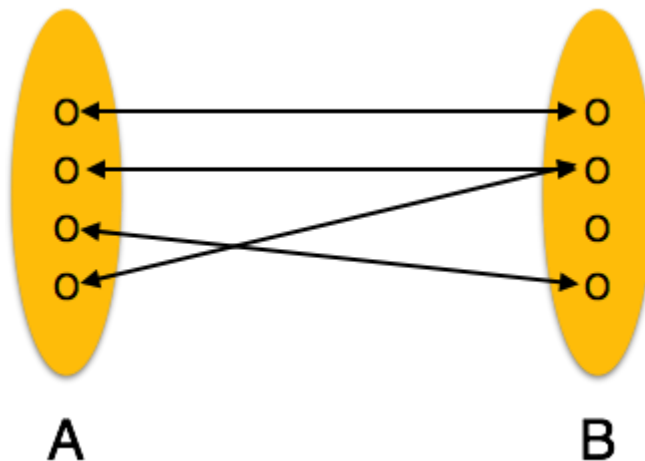
- ✓ **One-to-one** – One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



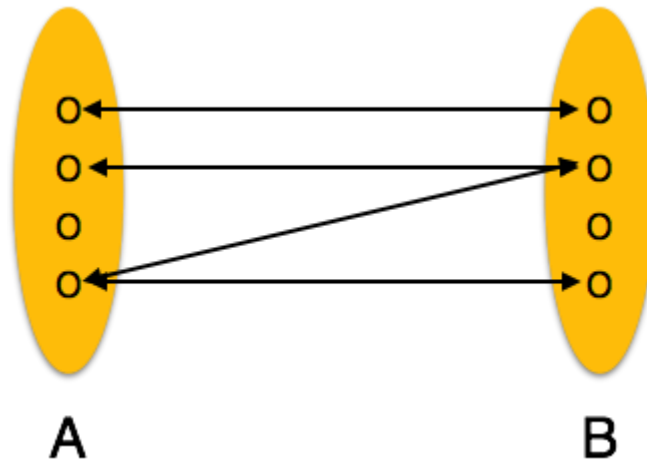
- ✓ **One-to-many** – One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



- ✓ **Many-to-one** – More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.



- ✓ **Many-to-many** – One entity from A can be associated with more than one entity from B and vice versa.



Keys

Keys are very important part of Relational database model. They are used to establish and identify relationships between tables and also to uniquely identify any record or row of data inside a table.

A Key can be a single attribute or a group of attributes, where the combination may act as a key.

Why we need a Key?

In real world applications, number of tables required for storing the data is huge, and the different tables are related to each other as well.

Also, tables store a lot of data in them. Tables generally extends to thousands of records stored in them, unsorted and unorganized.

Now to fetch any particular record from such dataset, you will have to apply some conditions, but what if there is duplicate data present and every time you try to fetch some data by applying certain condition, you get the wrong data. How many trials before you get the right data?

To avoid all this, **Keys** are defined to easily identify any row of data in a table.

Let's try to understand about all the keys using a simple example.

student_id	name	phone	age
1	Akon	9876723452	17
2	Akon	9991165674	19
3	Bkon	7898756543	18
4	Ckon	8987867898	19
5	Dkon	9990080080	17

Let's take a simple Student table, with fields **student_id**, **name**, **phone** and **age**.

Super Key

Super Key is defined as a set of attributes within a table that can uniquely identify each record within a table. Super Key is a superset of Candidate key.

In the table defined above super key would include `student_id`, `(student_id, name)`, `phone` etc.

Confused? The first one is pretty simple as `student_id` is unique for every row of data, hence it can be used to identity each row uniquely.

Next comes, `(student_id, name)`, now name of two students can be same, but their `student_id` can't be same hence this combination can also be a key.

Similarly, phone number for every student will be unique, hence again, `phone` can also be a key. So they all are super keys.

Candidate Key

Candidate keys are defined as the minimal set of fields which can uniquely identify each record in a table. It is an attribute or a set of attributes that can act as a Primary Key for

a table to uniquely identify each record in that table. There can be more than one candidate key.


In our example, `student_id` and `phone` both are candidate keys for table **Student**.

- A candidate key can never be NULL or empty. And its value should be unique.
- There can be more than one candidate keys for a table.
- A candidate key can be a combination of more than one columns(attributes).

Primary Key

Primary key is a candidate key that is most appropriate to become the main key for any table. It is a key that can uniquely identify each record in a table.

Primary Key for this table




student_id	name	age	phone

For the table **Student** we can make the `student_id` column as the primary key.

Composite Key

Key that consists of two or more attributes that uniquely identify any record in a table is called **Composite key**. But the attributes which together form the **Composite key** are not a key independently or individually.

Composite Key


student_id	subject_id	marks	exam_name

Score Table - To save scores of the student for various subjects.

In the above picture we have a **Score** table which stores the marks scored by a student in a particular subject.

In this table `student_id` and `subject_id` together will form the primary key, hence it is a composite key.

Cardinalities

The definition of cardinality that matters a lot for query performance is *data* cardinality. This is all about how many distinct values are in a column. The first meaning of cardinality is when you're designing the database—what's called data modeling. In this sense, cardinality means whether a relationship is one-to-one, many-to-one, or many-to-many. So you're really talking about the *relationship* cardinality.

Picture a product description table in an e-commerce database:

ProductID	Category	Name
788830	Headphones	Bose QuietComfort
313096	Bluetooth Speakers	Polk BOOM
814976	Bluetooth Speakers	JBL Clip
105945	Headphones	Audio-Technica ATH
666721	Bluetooth Speakers	Jamo DS3

The `ProductID` column is going to be high-cardinality because it's probably the primary key of that table, so it's totally unique. If there's a thousand rows in the table, there'll be a thousand different `ProductID` values. The `Category` column will have a lot of repetition, and it'll be low or medium cardinality: maybe 50 or 100 different `Category` values. `Name` is probably high cardinality, unless there's more to this table than meets the eye (such as multiple rows for different product colors and other variations).

Cardinality impacts performance a lot, because it influences the query execution plan. The planner will examine column statistics and use them to figure out how many values a query is likely to match, among other things. Depending on what it finds, it might use different query execution plans to try to get the best performance.

ER-Diagram

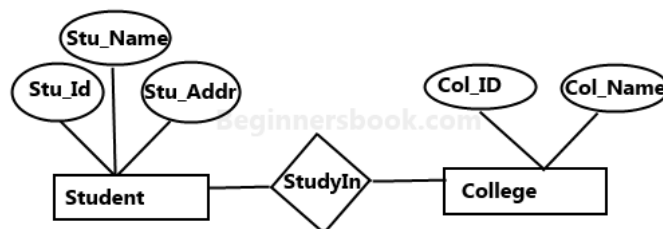
An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity 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. Let's have a look at a simple ER diagram to understand this concept.

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

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

1. Entity
2. Attribute
3. Relationship

1. Entity

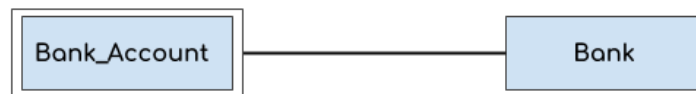
An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.

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.



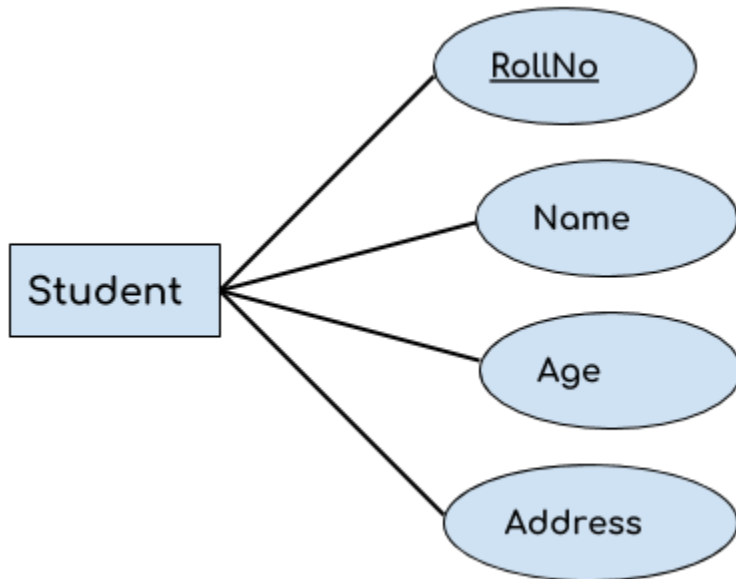
2. Attribute

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

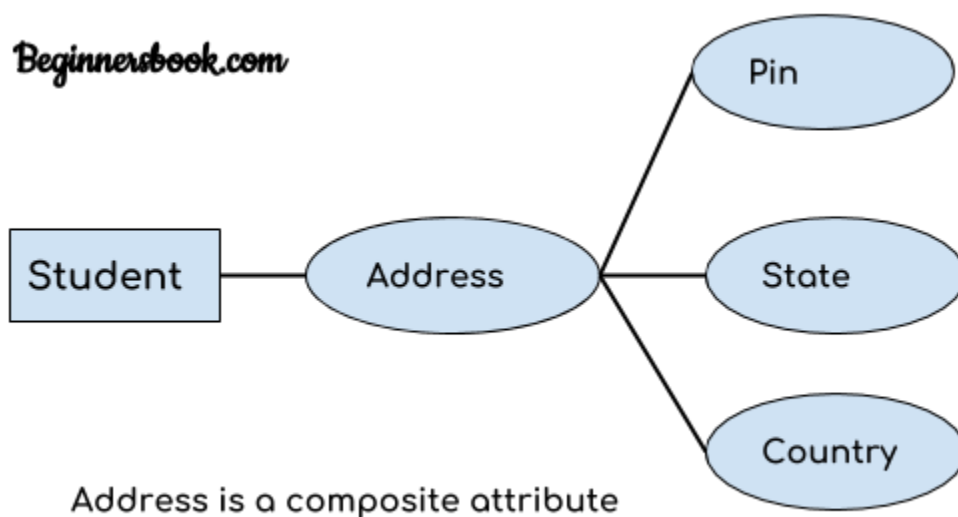
1. Key attribute:

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 attributes such as pin code, state, country.



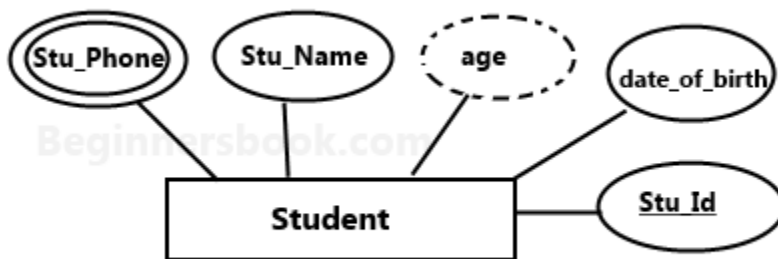
3. Multivalued attribute:

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:



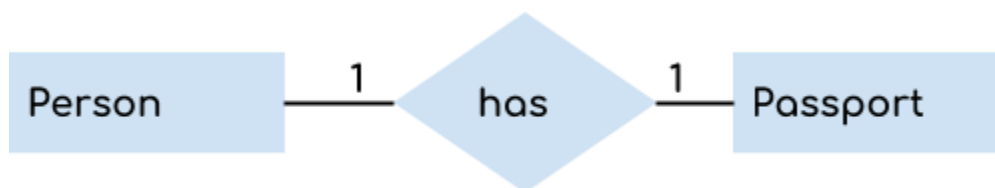
3. Relationship

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.



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 relationships. For example – a customer can place many orders but a order cannot be placed by many customers.



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.



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

Data Dictionary

A data dictionary contains metadata i.e. data about the database. The data dictionary is very important as it contains information such as what is in the database, who is allowed to access it, where is the database physically stored etc. The users of the database normally don't interact with the data dictionary; it is only handled by the database administrators.

The data dictionary in general contains information about the following:

1. Names of all the database tables and their schemas.
2. Details about all the tables in the database, such as their owners, their security constraints, when they were created etc.
3. Physical information about the tables such as where they are stored and how.
4. Table constraints such as primary key attributes, foreign key information etc.
5. Information about the database views that are visible.

Unit 3: Relation Model

3.1 Properties of relation

A relation, or table, in a relational database has certain properties. First off, its name must be unique in the database, i.e. a database cannot contain multiple tables of the same name. Next, each relation must have a set of columns or attributes, and it must have a set of rows to contain the data. As with the table names, no attributes can have the same name.

Next, no tuple (or row) can be a duplicate. In practice, a database might actually contain duplicate rows, but there should be practices in place to avoid this, such as the use of unique primary keys (next up).

Given that a tuple cannot be a duplicate, it follows that a relation must contain at least one attribute (or column) that identifies each tuple (or row) uniquely. This is usually the primary key. This primary key cannot be duplicated. This means that no tuple can have the same unique, primary key. The key cannot have a **NULL** value, which simply means that the value must be known.

Further, each cell, or field, must contain a single value. For example, you cannot enter something like "Sajal Shrestha" and expect the database to understand that you have a first and last name; rather, the database will understand that the value of that cell is exactly what has been entered.

Finally, all attributes—or columns—must be of the same domain, meaning that they must have the same data type. You cannot mix a string and a number in a single cell.

All these properties, or constraints, serve to ensure data integrity, important to maintain the accuracy of data.

3.2 Schemas

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 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. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.

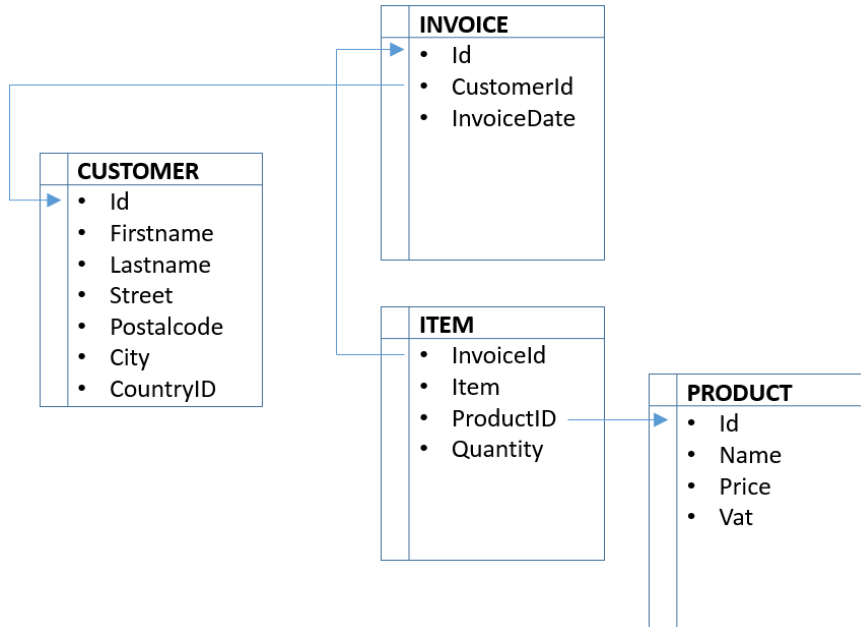
Tuples

In the context of relational databases, a tuple is one record (one row). The information in a database can be thought of as a spreadsheet, with columns (known as fields or attributes) representing different categories of information, and tuples (rows) representing all the information from each field associated with a single record.

Domains

A database domain, at its simplest, is the data type used by a column in a database. This data type can be a built-in type (such as an integer or a string) or a custom type that defines constraints on the data.

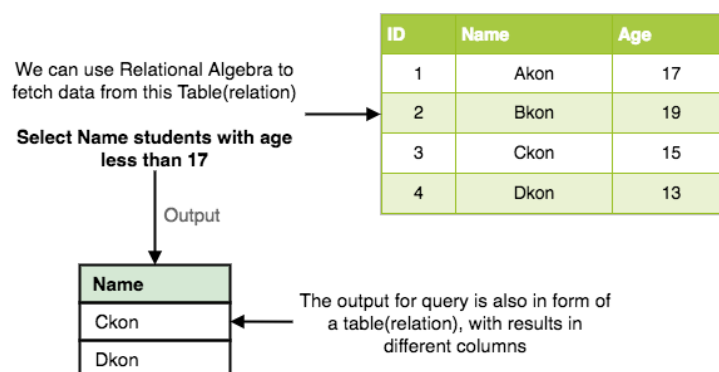
Schema Diagram



3.3 Relation Algebra

Every database management system must define a query language to allow users to access the data stored in the database. **Relational Algebra** is a procedural query language used to query the database tables to access data in different ways.

In relational algebra, input is a relation (table from which data has to be accessed) and output is also a relation (a temporary table holding the data asked for by the user).



Relational Algebra works on the whole table at once, so we do not have to use loops etc to iterate over all the rows(tuples) of data one by one. All we have to do is specify the

table name from which we need the data, and in a single line of command, relational algebra will traverse the entire given table to fetch data for you.

The primary operations that we can perform using relational algebra are:

1. Select
2. Project
3. Union
4. Set Different
5. Cartesian product
6. Rename

Select Operation (σ)

This is used to fetch rows(tuples) from table(relation) which satisfies a given condition

Project Operation (π)

Project operation is used to project only a certain set of attributes of a relation. In simple words, if you want to see only the **names** all of the students in the **Student** table, then you can use Project Operation.

It will only project or show the columns or attributes asked for, and will also remove duplicate data from the columns.

Union Operation (\cup)

This operation is used to fetch data from two relations (tables) or temporary relation (result of another operation).

For this operation to work, the relations(tables) specified should have same number of attributes(columns) and same attribute domain. Also the duplicate tuples are automatically eliminated from the result.

Set Difference ($-$)

This operation is used to find data present in one relation and not present in the second relation. This operation is also applicable on two relations, just like Union operation.

Cartesian Product (\times)

This is used to combine data from two different relations (tables) into one and fetch data from the combined relation.

Rename Operation (ρ)

This operation is used to rename the output relation for any query operation which returns result like Select, Project etc. Or to simply rename a relation (table)

Unit 4: SQL

4.1 Introduction to SQL, DDL and DML

SQL

SQL is a standard language for accessing and manipulating databases.

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

DDL (Data Definition Language)

Data Definition Language (DDL) is a standard for commands that define the different structures in a database. DDL statements create, modify, and remove database objects such as tables, indexes, and users. Common DDL statements are CREATE, ALTER, and DROP.

DML (Data Manipulation Language)

A data manipulation language (DML) is a family of computer languages including commands permitting users to manipulate data in a database. This manipulation involves inserting data into database tables, retrieving existing data, deleting data from existing tables and modifying existing data. DML is mostly incorporated in SQL databases.

4.2 Basic Structure of SQL Statement and SQL Queries

The SQL SELECT Statement

The SELECT statement is used to select data from a database. The data returned is stored in a result table, called the result-set.

SELECT Syntax

```
SELECT column1, column2, ...  
FROM table_name;
```

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

The SQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Syntax

```
SELECT DISTINCT column1, column2, ...  
FROM table_name;
```

Example

```
SELECT DISTINCT Country FROM Customers;
```

The SQL WHERE Clause

The WHERE clause is used to filter records. The WHERE clause is used to extract only those records that fulfill a specified condition.

WHERE Syntax

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

Note: The WHERE clause is not only used in SELECT statement, it is also used in UPDATE, DELETE statement, etc.!

Example

```
SELECT * FROM Customers  
WHERE Country='Mexico';
```

The SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The **AND** and **OR** operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Syntax

```
SELECT column1, column2, ...  
FROM table_name  
WHERE NOT condition;
```

Example of AND

```
SELECT * FROM Customers  
WHERE Country='Germany' AND City='Berlin';
```

Example of OR

```
SELECT * FROM Customers  
WHERE City='Berlin' OR City='München';
```

Example of NOT

```
ELECT * FROM Customers  
WHERE NOT Country='Germany';
```

The SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways.

The first way specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3, ...);
```

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT INTO syntax would be as follows:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

INSERT INTO Example

The following SQL statement inserts a new record in the "Customers" table:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City,
PostalCode, Country) VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen
21', 'Stavanger', '4006', 'Norway');
```

The SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2,           ...
WHERE condition;
```

Update Example

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

The SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

DELETE Syntax

```
DELETE FROM table_name WHERE condition;
```

Example

```
DELETE FROM Customers WHERE CustomerName='Sajal Shrestha';
```

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

Syntax

```
DELETE FROM table_name;
```

Example

```
DELETE FROM Customers;
```

Aggregate Functions

Aggregate functions are used to calculate results using field values from multiple records. There are five common aggregate functions.

The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

MIN() Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

MAX() Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

MIN() Example

The following SQL statement finds the price of the cheapest product:

Example

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

MAX() Example

The following SQL statement finds the price of the most expensive product:

Example

```
SELECT MAX(Price) AS LargestPrice  
FROM Products;
```

The SQL COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criteria.

The AVG() function returns the average value of a numeric column.

The SUM() function returns the total sum of a numeric column.

COUNT() Syntax

```
SELECT COUNT(column_name)  
FROM table_name  
WHERE condition;
```

Example

```
SELECT AVG(column_name)  
FROM table_name  
WHERE condition;
```

SUM() Syntax

```
SELECT SUM(column_name)  
FROM table_name  
WHERE condition;
```


Example

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

The SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards used in conjunction with the LIKE operator:

- % - The percent sign represents zero, one, or multiple characters
- _ - The underscore represents a single character

LIKE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

Here are some examples showing different LIKE operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%_ %'	Finds any values that start with "a" and are at least 3 characters in length

```
WHERE ContactName LIKE  
'a%o'
```

Finds any values that start with "a" and ends with "o"

SQL LIKE Examples

Example

```
SELECT * FROM Customers  
WHERE CustomerName LIKE 'a%';
```

The SQL CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a new SQL database.

Syntax

```
CREATE DATABASE databasename;
```

Example

```
CREATE DATABASE testDB;
```

SQL DROP DATABASE Statement

The DROP DATABASE statement is used to drop an existing SQL database

Syntax

```
DROP DATABASE databasename;
```

Example

```
DROP DATABASE testDB;
```

SQL CREATE TABLE Statement

The CREATE TABLE statement is used to create a new table in a database.

Syntax

```
CREATE TABLE table_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    ....  
);
```

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

SQL CREATE TABLE Example

The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

```
CREATE TABLE Persons (  
    PersonID int,  
    LastName varchar(255),  
    FirstName varchar(255),  
    Address varchar(255),  
    City varchar(255)  
);
```

The SQL DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

Example

```
DROP TABLE Person;
```

SQL ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name  
ADD column_name datatype;
```

The following SQL adds an "Email" column to the "Customers" table:

```
ALTER TABLE Customers  
ADD Email varchar(255);
```

ALTER TABLE - DROP COLUMN

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

Syntax

```
ALTER TABLE table_name  
DROP COLUMN column_name;
```

The following SQL deletes the "Email" column from the "Customers" table:

```
ALTER TABLE Customers
DROP COLUMN Email;
```

ALTER TABLE - ALTER/MODIFY COLUMN

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

SQL ALTER TABLE Example

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Now we want to add a column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

```
ALTER TABLE Persons
ADD DateOfBirth date;
```

RENAME Table Column

```
EXEC SP_RENAME 'table_name.old_name', 'new_name', 'COLUMN'
```

Note#: Remember to use single quotes to enclose your values.

SQL Create Constraints

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

Syntax

```
CREATE TABLE table_name (  
    column1 datatype constraint,  
    column2 datatype constraint,  
    column3 datatype constraint,  
    ....  
);
```

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

The following constraints are commonly used in SQL:

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified
- **INDEX** - Used to create and retrieve data from the database very quickly

SQL PRIMARY KEY on CREATE TABLE

```
CREATE TABLE Persons (  
    ID int NOT NULL PRIMARY KEY,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),
```

```
    Age int
);
```

4.4 Joined relation, Subqueries

Subqueries

Subqueries are queries embedded in queries. They are used to retrieve data from one table based on data in another table. They generally are used when tables have some kind of relationship. For example, in the Shop database, the **Orders** table has a **CustomerID** field, which references a customer in the **Customers** table. Retrieving the **CustomerID** for a specific order is pretty straightforward.

```
/*
Find the CustomerID of the company that placed order 10290.
*/
SELECT CustomerID
FROM Orders
WHERE OrderID = 10290;
```

This will return Customer ID, which is very likely meaningless to the people reading the report. The next query uses a subquery to return a meaningful result.

```
-- Find the name of the company that placed order 10290.

SELECT CompanyName
FROM Customers
WHERE CustomerID = (SELECT CustomerID
                    FROM Orders
                    WHERE OrderID = 10290);
```

4.5 Sets Operation

Set operators are basically used to combine the result of two or more SELECT statements. These set operations are performed on the data tables to get some kind of meaningful results from the data.

The different types of set operations performed on a database are, UNION, UNION ALL, INTERSECT.

I will use the following two tables to explain them one by one.

ID	NAME
3	Deadpool
1	Batman

ID	NAME
1	Batman
2	Superman

UNION: The union operator will combine the results of two select statements eliminating all the duplicate rows. So when we perform the following query

```
select * from first_table UNION select * from second table
```

we get the result as:

ID	ID
1	Batman
2	Superman
3	Deadpool

UNION ALL: Union all is similar to Union, except it also shows the duplicate records.

*select * from first_table UNION ALL select * from second table;*

ID	ID
1	Batman
2	Superman
3	Deadpool
1	Batman

INTERSECT: Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements.

*select * from first_table INTERSECT select * from second table;*

ID	ID
1	Batman

MINUS: Minus operation combines result of two Select statements and return only those result which belongs to first set of result. MySQL does not support INTERSECT operator.

*select * from first_table MINUS select * from second table;*

ID	ID
2	Superman

4.6 Views

A VIEW in SQL is a logical subset of data from one or more tables. View is used to restrict data access.

Syntax for creating a View,

```
CREATE or REPLACE VIEW view_name
AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

As you may have understood by seeing the above SQL query, a view is created using data fetched from some other table(s). It's more like a temporary table created with data.

Creating a VIEW

Consider following **Sale** table,

oid	order_name	previous_balance	customer
11	ord1	2000	Alex
12	ord2	1000	Adam
13	ord3	2000	Abhi
14	ord4	1000	Adam
15	ord5	2000	Alex

SQL Query to Create a View from the above table will be,

```
CREATE or REPLACE VIEW sale_view  
AS  
SELECT * FROM Sale WHERE customer = 'Alex';
```

The data fetched from **SELECT** statement will be stored in another object called **sale_view**. We can use **CREATE** and **REPLACE** separately too, but using both together works better, as if any view with the specified name exists, this query will replace it with fresh data.

Displaying a VIEW

The syntax for displaying the data in a view is similar to fetching data from a table using a **SELECT** statement.

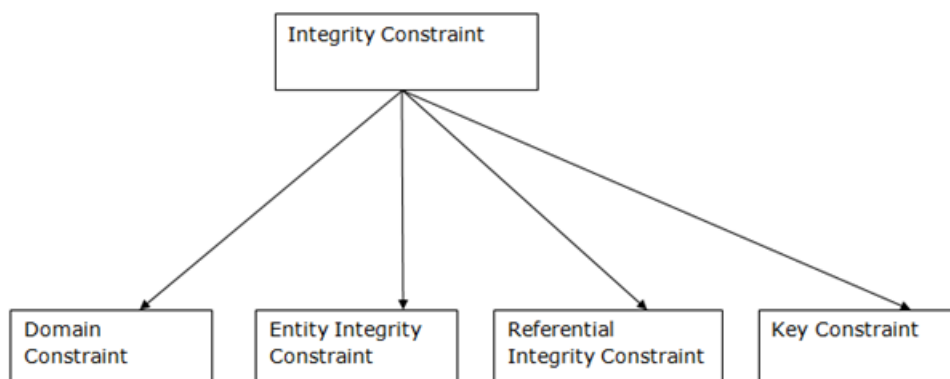
```
SELECT * FROM sale_view;
```

Unit 5 Integrity Constraints

Integrity Constraints

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

Types of Integrity Constraint



1.Domain constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example

ID	NAME	SEMENSTER	AGE
1000	Tom	1 st	17
1001	Johnson	2 nd	24
1002	Leonardo	5 th	21
1003	Kate	3 rd	19
1004	Morgan	8 th	A

Not allowed. Because AGE is an integer attribute

2.Entity integrity constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field

Example

EMPLOYEE

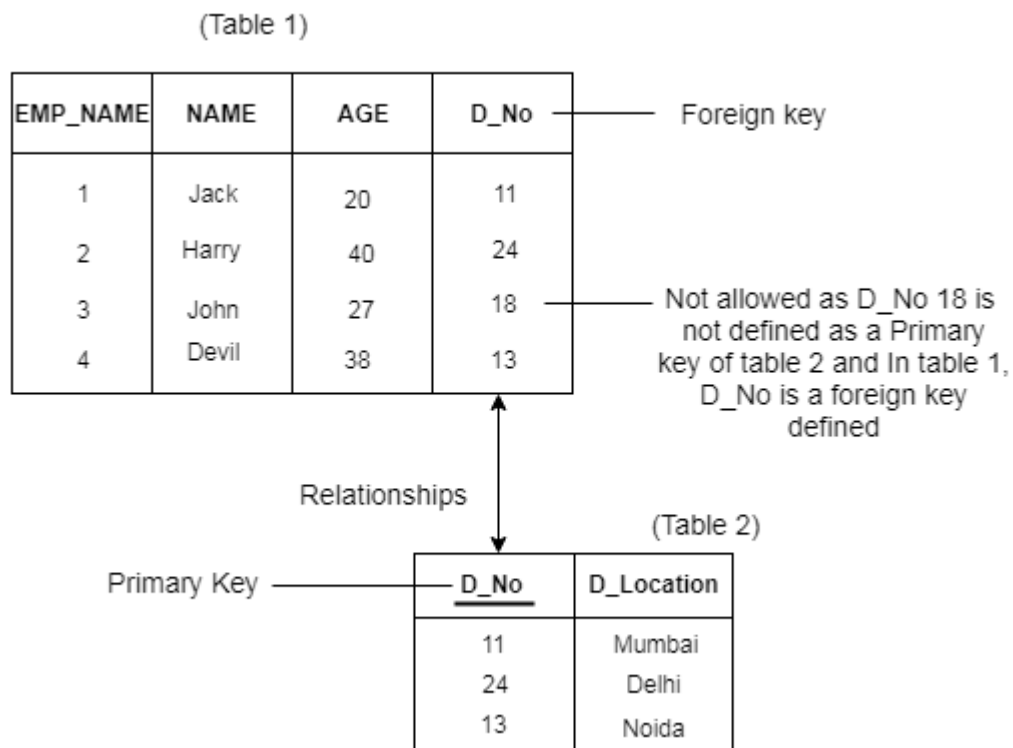
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example



4. Key constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.
- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

Example

ID	NAME	SEMENSTER	AGE
1000	Tom	1 st	17
1001	Johnson	2 nd	24
1002	Leonardo	5 th	21
1003	Kate	3 rd	19
1002	Morgan	8 th	22

Not allowed. Because all row must be unique

Unit VI: Relational Database Design

6.1 Pitfall of Relational Model

A bad design may have several properties, including:

- Repetition of information.
- Inability to represent certain information.
- Loss of information.

Data Redundancy

Data redundancy occurs when the same piece of data is held within a database or data storage technology. Data can reoccur in two different fields within a single database, or two different spots in multiple software platforms or environments. Data redundancy can arise by accident, but can also be done deliberately for backup and recovery purposes.

It is fine for data to be stored in multiple places; in order to avoid problems, it's important to have a central, master field or space for this data, so that there is a way to update all of the places where data is redundant through a single access point. Otherwise, this type of data redundancy can lead to data inconsistency, where one update does not automatically update another field. Pieces of data that are supposed to be identical end up having different values, which can lead to problems with processing.

Anomalies

Anomalies are problems that can occur in poorly planned, un-normalised databases where all the data is stored in one table (a flat-file database).

Insertion Anomaly - The nature of a database may be such that it is not possible to add a required piece of data unless another piece of unavailable data is also added. E.g. A library database that cannot store the details of a new member until that member has taken out a book.

Deletion Anomaly - A record of data can legally be deleted from a database, and the deletion can result in the deletion of the only instance of other, required data, E.g. Deleting

a book loan from a library member can remove all details of the particular book from the database such as the author, book title etc.

Modification Anomaly - Incorrect data may have to be changed, which could involve many records having to be changed, leading to the possibility of some changes being made incorrectly.

6.2 Functional Dependency

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 $\text{Stu_ID} \rightarrow \text{Stu_Name}$ 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 be represented as $A \rightarrow B$ (Attribute B is functionally dependent on attribute A)

6.3 Normalization, Its need and objective

Basically, normalization is the **process of efficiently organising data** in a database. There are two main objectives of the normalization process: **eliminate redundant data** (storing the same data in more than one table) and **ensure data dependencies make sense** (only storing related data in a table). Both of these are valuable goals as they **reduce the amount of space a database consumes** and **ensure that data is logically stored**.

6.4 1NF,2NF,3NF,BCNF and 4NF

First Normal Form (1NF): For a table to be in the First Normal Form, it should follow the following 4 rules:

1. It should only have single(atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain
3. All the columns in a table should have unique names.
4. And the order in which data is stored, does not matter.

Second Normal Form (2NF): For a table to be in the Second Normal Form,

1. It should be in the First Normal form.
2. And, it should not have Partial Dependency.

Third Normal Form (3NF): A table is said to be in the Third Normal Form when,

1. It is in the Second Normal form.
2. And, it doesn't have Transitive Dependency.

Boyce and Codd Normal Form (BCNF):

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

1. R must be in 3rd Normal Form
2. and, for each functional dependency ($X \rightarrow Y$), X should be a super Key.

Fourth Normal Form (4NF): A table is said to be in the Fourth Normal Form when,

1. It is in the Boyce-Codd Normal Form.
2. And, it doesn't have Multi-Valued Dependency.

Unit VII: Database Security

7.1 Access Control

One of the key foundations of a comprehensive IT security strategy involves implementing an appropriate level of access control to all computer systems in an organization or enterprise.

The term *Access Control* actually refers to the control over access to system resources *after* a user's account credentials and identity have been authenticated and access to the system granted. For example, a particular user, or group of users, might only be permitted access to certain files after logging into a system, while simultaneously being denied access to all other resources.

Discretionary and Mandatory Access Control

Mandatory Access Control (MAC) is a kind of access control that is based on asset classification e.g Top Secret, Secret, confidential etc... This is usually obtainable in high security organizations e.g Military etc

Discretionary Access Control (DAC) on the other hand is a type access control that is determined by the owner of an asset. e.g a user that creates a file determines who has read, write or execute permissions on the file

7.2 Authorization and Authentication

Authentication

Authentication is about validating your credentials like User Name/User ID and password to verify your identity. The system determines whether you are what you say you are using your credentials. In public and private networks, the system authenticates the user identity via login passwords. Authentication is usually done by a username and password, and sometimes in conjunction with factors of authentication, which refers to the various ways to be authenticated.

Authentication factors determine the various elements the system use to verify one's identity prior to granting him access to anything from accessing a file to requesting a bank transaction. A user's identity can be determined by what he knows, what he has, or what he is. When it comes to security, at least two or all the three authentication factors must be verified in order to grant someone access to the system

For example, when you enter your ATM card into the ATM machine, the machine asks you to enter your pin. After you enter the pin correctly, the bank then confirms your identity that the card really belongs to you and you're the rightful owner of the card. By validating your ATM card pin, the bank actually verifies your identity, which is called authentication. It merely identifies who you are, nothing else.

Authorization

Authorization, on the other hand, occurs after your identity is successfully authenticated by the system, which ultimately gives you full permission to access the resources such as information, files, databases, funds, locations, almost anything. In simple terms, authorization determines your ability to access the system and up to what extent. Once your identity is verified by the system after successful authentication, you are then authorized to access the resources of the system.

Authorization is the process to determine whether the authenticated user has access to the particular resources. It verifies your rights to grant you access to resources such as information, databases, files, etc. Authorization usually comes after authentication which confirms your privileges to perform. In simple terms, it's like giving someone official permission to do something or anything.

For example, the process of verifying and confirming employees ID and passwords in an organization is called authentication, but determining which employee has access to which floor is called authorization.

7.3 Data Encryption and Decryption

Database encryption is the process of converting data, within a database, in plain text format into a meaningless cipher text by means of a suitable algorithm.

Database decryption is converting the meaningless cipher text into the original information using keys generated by the encryption algorithms.

Unit VIII: Transaction Management, Recovery, and Query Processing

8.1 Introduction to Transaction

Transaction - The *execution* of a program that performs an administrative function by accessing a *shared database*, usually on behalf of an *on-line* user.

Transaction Processing – The technology for building large-scale systems that run transactions

Example

- Reserve an airline seat.
- Buy an airline ticket
- Get cash from an ATM.
- Download a video clip

ACID Properties

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 **Atomicity**, **Consistency**, **Isolation**, and **Durability** – commonly known as ACID properties

Atomicity: A transaction must be fully complete, saved (committed) or completely undone (rolled back). A sale in a retail store database illustrates a scenario which explains atomicity, e.g., the sale consists of an inventory reduction and a record of incoming cash. Both either happen together or do not happen - it's all or nothing.

Consistency: The transaction must be fully compliant with the state of the database as it was prior to the transaction. In other words, the transaction cannot break the database's constraints. For example, if a database table's Phone Number column can only contain

numerals, then consistency dictates that any transaction attempting to enter an alphabetical letter may not commit.

Isolation: Transaction data must not be available to other transactions until the original transaction is committed or rolled back.

Durability: Transaction data changes must be available, even in the event of database failure.

8.2 Introduction to concurrency control

Concurrency control is a database management systems (DBMS) concept that is used to address conflicts with the simultaneous accessing or altering of data that can occur with a multi-user system. concurrency control, when applied to a DBMS, is meant to coordinate simultaneous transactions while preserving data integrity.

Example

To illustrate the concept of concurrency control, consider two travelers who go to electronic kiosks at the same time to purchase a train ticket to the same destination on the same train. There's only one seat left in the coach, but without concurrency control, it's possible that both travelers will end up purchasing a ticket for that one seat. However, with concurrency control, the database wouldn't allow this to happen. Both travelers would still be able to access the train seating database, but concurrency control would preserve data accuracy and allow only one traveler to purchase the seat.

8.3 Reason for transaction failure

A transaction has to abort when it fails to execute or when it reaches a point from where it can't go any further. This is called transaction failure where only a few transactions or processes are hurt.

Reasons for a transaction failure could be –

Logical errors – Where a transaction cannot complete because it has some code error or any internal error condition.

System errors – Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.

System Crash

There are problems – external to the system – that may cause the system to stop abruptly and cause the system to crash. For example, interruptions in power supply may cause the failure of underlying hardware or software failure.

Examples may include operating system errors.

Disk Failure

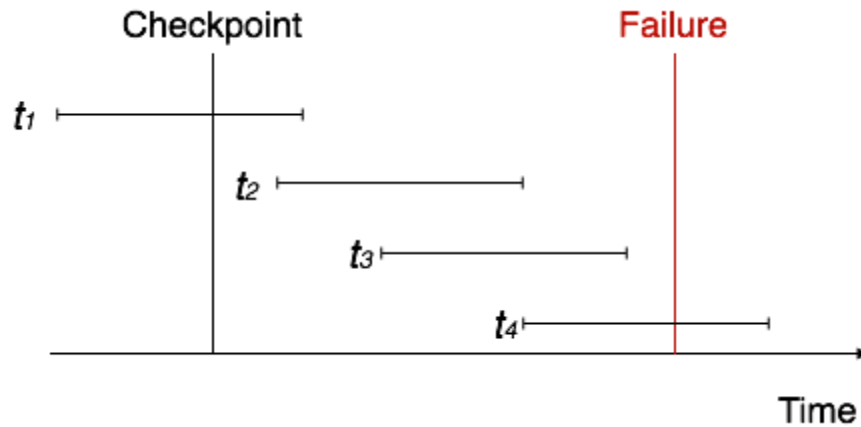
In early days of technology evolution, it was a common problem where hard-disk drives or storage drives used to fail frequently.

Disk failures include formation of bad sectors, unreachability to the disk, disk head crash or any other failure, which destroys all or a part of disk storage.

System Recovery and Media Recovery

DBMS is a highly complex system with hundreds of transactions being executed every second. The durability and robustness of a DBMS depends on its complex architecture and its underlying hardware and system software. If it fails or crashes amid transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.

When a system with concurrent transactions crashes and recovers, it behaves in the following manner –



- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an undo-list and a redo-list.
- If the recovery system sees a log with $\langle T_n, \text{Start} \rangle$ and $\langle T_n, \text{Commit} \rangle$ or just $\langle T_n, \text{Commit} \rangle$, it puts the transaction in the redo-list.
- If the recovery system sees a log with $\langle T_n, \text{Start} \rangle$ but no commit or abort log found, it puts the transaction in undo-list.

8.4 Introduction to query processing

- Query Processing is a translation of high-level queries into low-level expression.
- It is a step wise process that can be used at the physical level of the file system, query optimization and actual execution of the query to get the result.
- It requires the basic concepts of relational algebra and file structure.
- It refers to the range of activities that are involved in extracting data from the database.
- It includes translation of queries in high-level database languages into expressions that can be implemented at the physical level of the file system.
- In query processing, we will actually understand how these queries are processed and how they are optimized.

Step in Query Processing

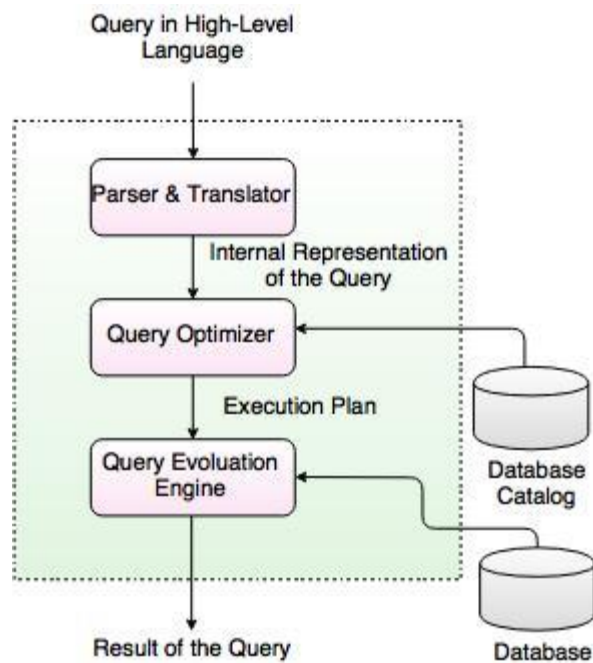


Fig. Query Processing

In the above diagram,

- The first step is to transform the query into a standard form.
- A query is translated into SQL and into a relational algebraic expression. During this process, Parser checks the syntax and verifies the relations and the attributes which are used in the query.
- The second step is Query Optimizer. In this, it transforms the query into equivalent expressions that are more efficient to execute.
- The third step is Query evaluation. It executes the above query execution plan and returns the result.

