RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL

New Scheme Based On AICTE Flexible Curricula

Information Technology, IV-Semester

IT405 - Data Base Management System

Course Objectives:

The main objectives of the course are
1. To understand fundamental knowledge of file system, database concepts and use of relational database.
2. To study of different data model and conceptual design using ER diagram.
3. Students can use SQL operations to manipulate the database and learn how to design and create a good database using functionaldependencies and normalization.
4. The course provides an overview of transaction management, concurrency control, distributed database and Big Data.

Basic Concepts: Introduction to DBMS, File system vs DBMS, Advantages of database systems, Database System architecture, Data models, Schemas and instances, Data independence, Functions of DBA and designer, Entities and attributes, Entity types, Key attributes, Relationships, Defining the E-R diagram of database.

Relational Model: Structure of relational databases, Domains, Relations, Relational algebra – fundamental operators and syntax, relational algebra queries, Entity-Relationship model :Basic concepts, Design process, constraints, Keys, Design issues, E-R diagrams, weak entity sets, extended E-R features –generalization, specialization and aggregation

SQL: Data definition in SQL, update statements and views in SQL: Data storage and definitions, Data retrieval queries and update statements, Query Processing & Query Optimization: Overview, measures of query cost, selection operation, sorting, join, evaluation of expressions, transformation of relational expressions, estimating statistics of expression results, evaluation plans. Case Study of ORACLE and DB2.

Relational Database design: Functional Dependency –definition, trivial and non-trivial FD, closure of FD set, closure of attributes, irreducible set of FD, Normalization –1NF, 2NF, 3NF, Decomposition using FD-dependency preservation, lossless join, BCNF, Multi-valued dependency, 4NF, Join dependency and 5NF

Introduction of transaction, transaction processing and recovery, Concurrency control: Lock management, specialized locking techniques, concurrency control without locking, Protection and Security Introduction to: Distributed databases, Basic concepts of object oriented data base system.

Course Outcomes:
After successful completion of this course, the students would be able to:
1. Compare file system and DBMS and explain how DBMS is better than traditional File Processing Systems.

2. Analyze the physical and logical database designs, database modeling, relational, hierarchical, and network models

3. Analyze and renovate an information model into a relational database schema and to use a DDL, DML and DCL utilities to implement the schema using a DBMS.
4. Formulate data retrieval queries in SQL and Relational Algebra.
5. Demonstrate an understanding of functional dependencies, normalization theory and apply such knowledge to the design of adatabase.
6. Demonstrate and explain terms like Transaction Processing, Concurrency Control, distributed database and big data.

Reference Books:

1. Korth, Silbertz, Sudarshan, "Database Concepts", McGraw Hill.
2. Elmasri, Navathe, "Fundamentals of Database Systems", Pearson.
3. Ivan Bayross, "SQL, PL/SQL the Programming Language of Oracle", BPB publications.
4. S. Sharma, J. Agrawal, S. Agrawal, "Advanced Database Management System", Dreamtech Press.
5. Leon & Leon, "Fundamental of Data Base Management System", TMH

List of Experiments:

1. To perform various SQL Commands of DDL, DML, DCL.
2. Write SQL Commands such as Insertion, deletion and updation for anyschema.
3. To execute Nested Queries, Join Queries, order-by, having clause and stringoperation.
4. To perform set operators like Union, Intersect, Minus on a set of tables.
5. To execute various commands for GROUP functions (avg, count, max, min, Sum).
6. Write a PL/SQL block for transaction application using Triggers.
7. Write a DBMS program to prepare report for an application using function.
8. Designing of various Input screens/Forms.
9. Create reports using database connectivity of Front end with back end.
10. Create database Design with normalization and implementing in anyapplication.

Class Notes

SEMESTER: -IT-IV SEM **Subject Name: -DBMS**

UNIT -I

Introduction

The database is a collection of similar data. The database management system is software designed to assist the maintenance and utilisation of large-scale collection of data. DBMS came into existence in 1960 by Charles. Integrated data store which is also called as the first general-purpose DBMS. Again in 1960 IBM brought IMS-Information management system. In 1970 Edgar Codd at IBM came with a new database called RDBMS. In 1980 then came SQL Architecture- Structure Query Language. From 1980 to 1990 there were advances in DBMS, e.g. DB2, ORACLE.

Data

• Data is raw fact or figures or entity.
• When activities in the organisation take place, the effect of these activities needs to be recorded which is known as Data.

Information

Processed data is called information
The purpose of data processing is to generate the information required for carrying out business activities.

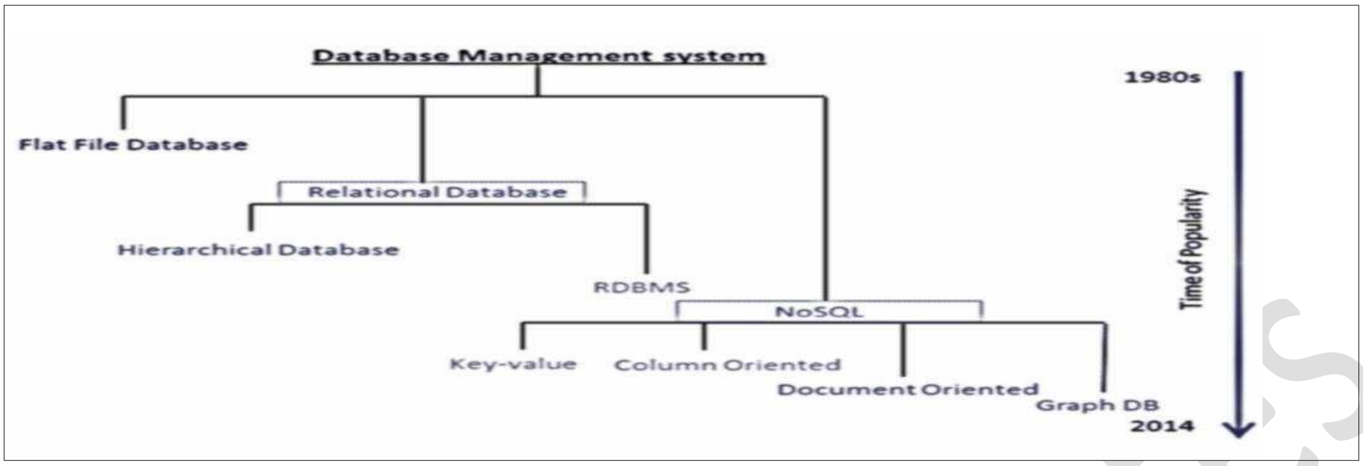
We can further define different functions of DBMS in details like.

- **Data capture**: Collection of data from the place of origination like taking attendance in the class using at PDA.
- **Data classification:** Categorization of captured data based on different dimensions like size, type like segregation of data images, audio, video.
- **Data storage:** The categorised data must be stored, and it must maintain data persistence and integrity of it like we are storing details of students and it should remain the same for years.
- **Data arranging:** Arrangement of the data in the database in a proper manner that will help the user to understand how we are going to use it.
- **Data Retrieval:** Data required for retrieval so there must be mechanics through which they can be retrieved efficiently. All the DBMS software provider that will support a familiar language SQL used for the retrieval of data.
- **Data maintenance:** It is the database to keep it up-to-date. One of the critical feature of the DBMS through which data can be updated and remain useful for the user.
- **Data Verification:** Before storing the data, it must be checked based on the syntax and semantics to avoid errors. To keep data persistence, it is mandatory to verify it before the storage and must store in the database the way it supports.
- **Data Editing:** Providing tools that facilitate to perform change and update operation on DBMS.
- **Data transcription:** Providing a facility to transfer one form to another. User requirement changes very frequently and in the digital world expectations from the users are also very high. So, DBMS must provide facility to change it from one form to other.

Database

Let us understand first about the what is a database. So, a database is a set of data, and it contains interrelated data. The database contains a set of algorithm and rules through which data can be stored in it systematically. So, all the data fields must be related to each other. For example, where it is used suppose an organisation from attendance, salary calculation and different allowances are given to them can be done through the Database. To keep a record nowadays from billing an item to take attendance is the classroom database is required.

Evolution of DBMS: -



File Oriented Approach: -

Computer System comes into the picture to store the records and after performing computation producing information. As compared to the manual work done by the human being, they are more accurate and faster and secure also. Such a system stores a group of records like department wise or class wise record and each group has separate files, or for a category. Such arrangements called file processing system and in which each branch or department is having its file and a dedicated application designed to operate it. This is mainly to keep a record of the students like fees details, result details, and contact details and whenever some information is required then through an application program, it can be accessed. Still, file processing approach of data storage and access is used, but it has some drawbacks.

Database Management System

A Database Management System (DBMS) is a collection of programs that enables the user to create and maintain a database.
The DBMS is hence a general-purpose software system that facilitates the process of defining constructing and manipulating database for various applications.
Comparison Between File System & DBMS

DBMS	File System
1. DBMS is a collection of data and user is not required to write the procedures for managing the database. 2. DBMS provides an abstract view of data that hides the details. 3. DBMS is efficient to use since there are wide varieties of sophisticated techniques to store and retrieve the data. 4. DBMS takes care of Concurrent access using some form of locking. 5. DBMS has crash recovery mechanism DBMS protects the user from the effects of system failures. 6. DBMS has a good protection mechanism.	1. The file system is a collection of data. Any management with the file system, the user must write the procedures 2. File system gives the details of the data representation and Storage of data. 3. In File system storing and retrieving of data cannot be done efficiently. 4. Concurrent access to the data in the file system has many problems like a. Reading the file while other deleting some information, updating some information 5. The file system does not provide a crash recovery mechanism. E.g. While we are entering some data into the file if System crashes then the content of the file is lost. 6. Protecting a file under file system is very difficult.

A database management system is, well, a system used to manage databases.

RDBMS
A relational database management system is a database management system used to manage relational databases. A relational database is one where tables of data can have
relationships based on primary and foreign keys.

Advantages of DBMS
Due to its centralised nature, the database system can overcome the disadvantages of the file system-based system

- **Concurrent Use** a database system provides facility to access database several users concurrently. Let we understand it by taking an example that a movie online booking system database employee of different branches concurrently access the database. Each employee can handle customers at their desk, Able to see the seats available for the booking from the interface provided to them.
- **Structured data & details** one of the essential features of the database system is not only to store the data and to provide access to the database. However, it also provides details about the data and how to access and use it. Taking an example when you are going for an online form of exam then details about every field is given, what type of data it accepts and format details also.
- **Data Independence**, another essential characteristic of the DBMS, is data independence in which application through which user can interact with the user is not dependent on the physical data storage. So, if there is any change in the application program, it will not affect the data stored in the database.
- **The integrity of data** This characteristic of the DBMS deals with one of the fundamental properties of the data called integrity, in which if some data is saved in the database and later retrieved from the database it must be same. This also covers restriction on the unauthorised access of the data that can make changes in the data sets.
- **Transaction of Data,** a transaction is a set of actions that are done in a database to transfer it from one consistent state to another. If it is not handled correctly, it may result in inconsistent state and loss of data, so DBMS has certain constraints to maintain the fundamental properties of transaction like atomicity, integrity, isolation, and durability.
- **Data Persistence,** this is one of the essential characteristics of the database because data can be retained in the database for years and it should be in the same condition. In the banking system, a user will open his account and lifelong maintain it, or a user has opted an LIC policy, or media claim policy.

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 behaviour 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.
Isolation of data and application − A database system is entirely different from its data. A database is an active entity, whereas data is said to be passive, on which the database works and organises. DBMS also stores metadata, which is data about data, to ease its process.
Less redundancy − DBMS follows the rules of normalisation, which splits a relation when any of its attributes is having redundancy in values. Normalisation is a mathematically rich and scientific process that reduces data redundancy.
Consistency − is a state where every relation in a database remains consistent. There exist methods and techniques, which can detect an attempt of leaving the database in the inconsistent state.
Query Language − DBMS is equipped with a 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 the file-processing system was used.
ACID Properties − DBMS follows the concepts of Atomicity, Consistency, Isolation, and Durability (commonly

shortened as ACID). These concepts are applied to transactions, which manipulate data in a database.
Multiuser 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, 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 a database than a person working in the Production department. This feature enables the users to have a concentrated 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 from 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 Architecture: -

The design of a DBMS depends on its architecture. It can be centralised or decentralised or hierarchical. The architecture of a DBMS be either single tier or multi-tier. An n-tier architecture divides the entire system into related but independent **n** modules, which can be independently modified, altered, changed, or replaced.

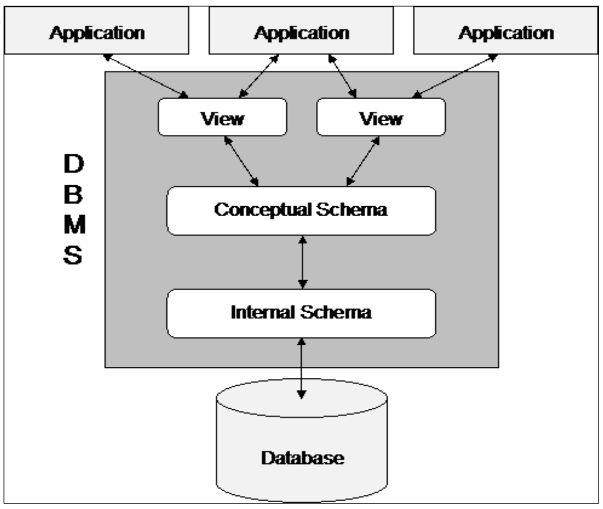
In 1-tier architecture, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-users. Database designers and programmers usually prefer to use single-tier architecture.

If the architecture of DBMS is 2-tier, then it must have an application through which the DBMS can be accessed. Programmers use a 2-tier architecture where they access the DBMS using an application. Here the application tier is entirely independent of the database regarding operation, design, and programming.

3-Tier Architecture
Database (Data) Tier − At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.

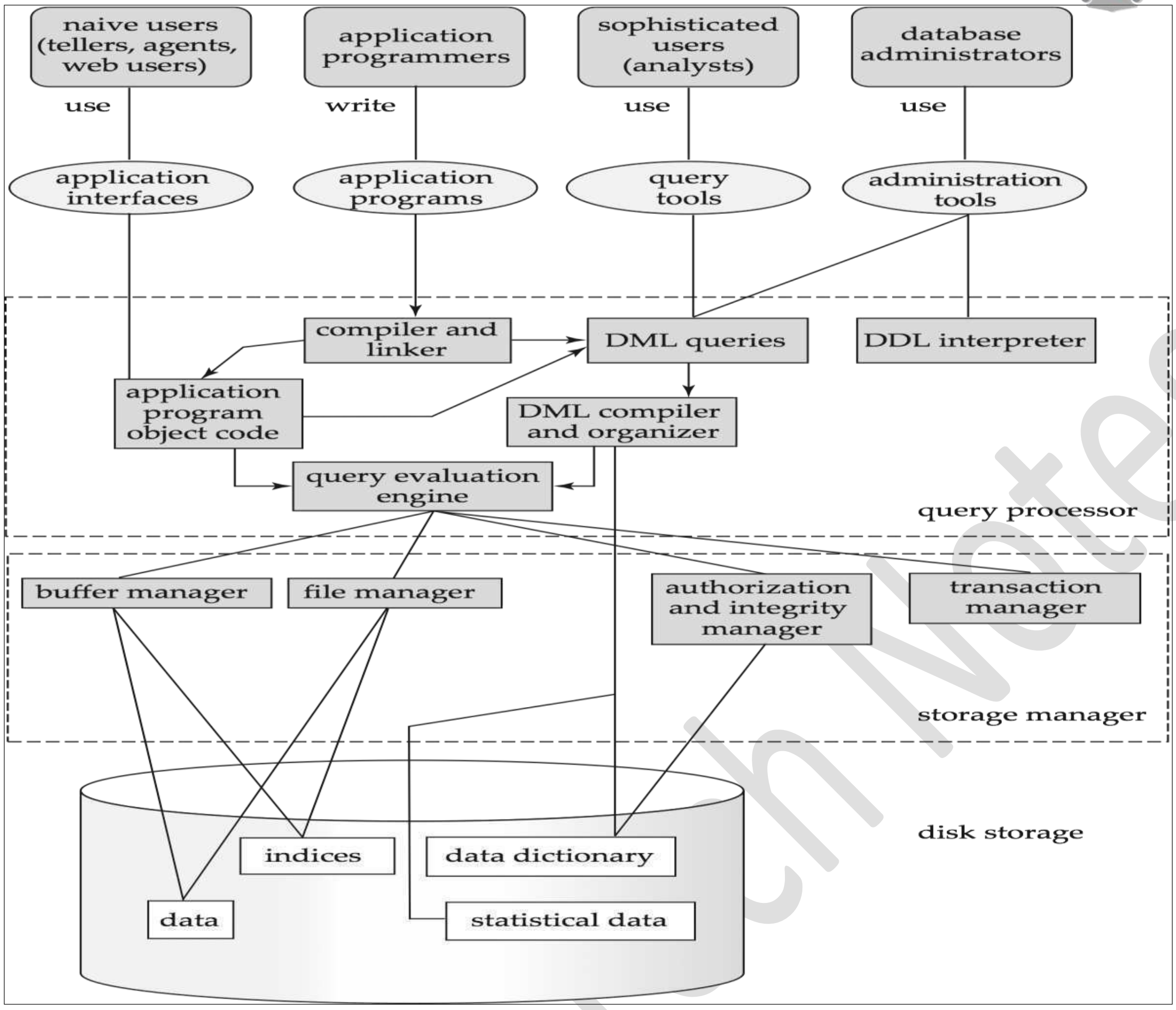
Application (Middle) Tier − At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.

User (Presentation) Tier − End-users operate on this tier, and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.



DBMS Three-Tier Architecture

DBMS Architecture Component: -



DBMS Architecture Component

Database Users:

Users are differentiated by the way they expect to interact with the system:

Application programmers:
Application programmers are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces.
Rapid application development (RAD) tools are tools that enable an application programmer to construct forms and reports without writing a program.
Sophisticated users:
Sophisticated users interact with the system without writing programs. Instead, they form their requests in a database query language.
Specialised users:
Specialised users are sophisticated users who write specialised database applications that do not fit into the traditional data-processing framework.
Naïve users:

Naive users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.
Database Administrator:
Coordinates all the activities of the database system. The database administrator has a good understanding of the enterprise's information resources and needs.

Query Processor:
The query processor will accept a query from a user and solves it by accessing the database.
Parts of Query processor:
DDL interpreter
This will interpret DDL statements and fetch the definitions in the data dictionary.
DML compiler
a. This will translates DML statements in a query language into low-level instructions that the query evaluation engine understands.
b. A query can usually be translated into any of some alternative evaluation plans for same query result DML compiler will select the best plan for query optimisation.
Query evaluation engine
This engine will execute low-level instructions generated by the DML compiler on DBMS.
Storage Manager/Storage Management:
A storage manager is a program module which acts as an interface between the data stored in a database and the application programs and queries submitted to the system.
The storage manager components include:
Authorisation and integrity manager: Checks for integrity constraints and authority of users to access data.
Transaction manager: Ensures that the database remains in a consistent state although there are system failures.
File manager: Manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
Buffer manager: It is responsible for retrieving data from disk storage into main memory. It enables the database to handle data sizes that are much larger than the size of main memory.

Data files: Stored in the database itself.
Data dictionary: Stores metadata about the structure of the database.
Indices: Provide fast access to data items.
Data Models
Data models define how the logical structure of a database is modelled. Data Models are fundamental entities to introduce abstraction in a DBMS. Data models define how data is connected and how they are processed and stored inside the system.

The very first data model could be flat data models, where all the data used are to be kept in the same plane. Earlier data models were not so scientific; hence they were prone to introduce lots of duplication and update anomalies.

There are many kinds of data models. Some of the most common ones include:

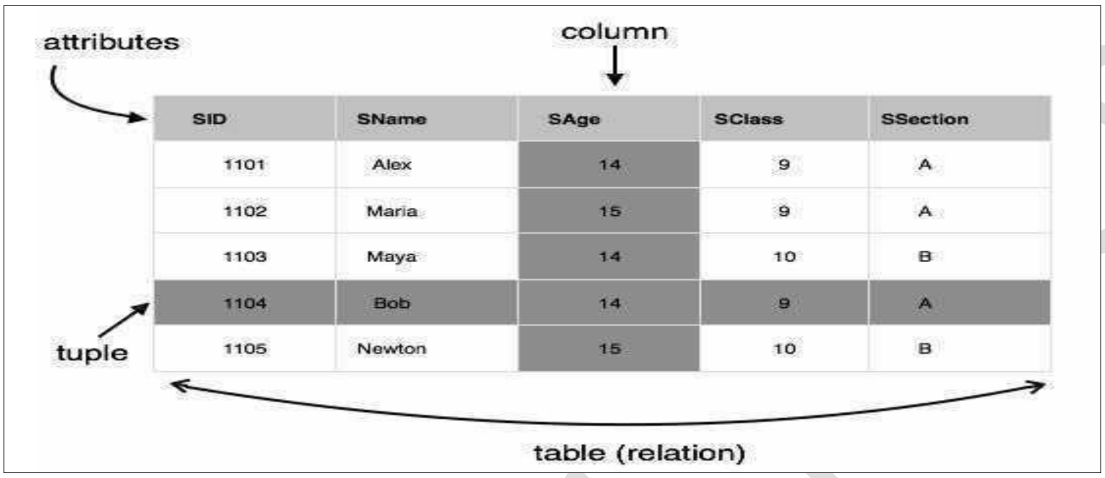
- Hierarchical database model
- Relational model
- Network model
- Object-oriented database model
- Entity-relationship model
- Document model
- Entity-attribute-value model
- Star Schema
- The object-relational model, which combines the two that make up its name.

The most common model, the relational model sorts data into tables, also known as relations, each of which consists of columns and rows. Each column lists an attribute of the entity in question, such as price, zip code, or birth date. Together, the attributes in a relation are called a domain. An attribute or combination of attributes is chosen as a primary key that can be referred to in other tables when it is called a foreign key.

Each row also called a tuple, includes data about a specific instance of the entity in question, such as an employee.

The model also accounts for the types of relationships between those tables, including one-to-one, one-to-many, and many-to-many relationships. Here's an example:

Relational model: -


attributes
column
tuple
table (relation)

SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

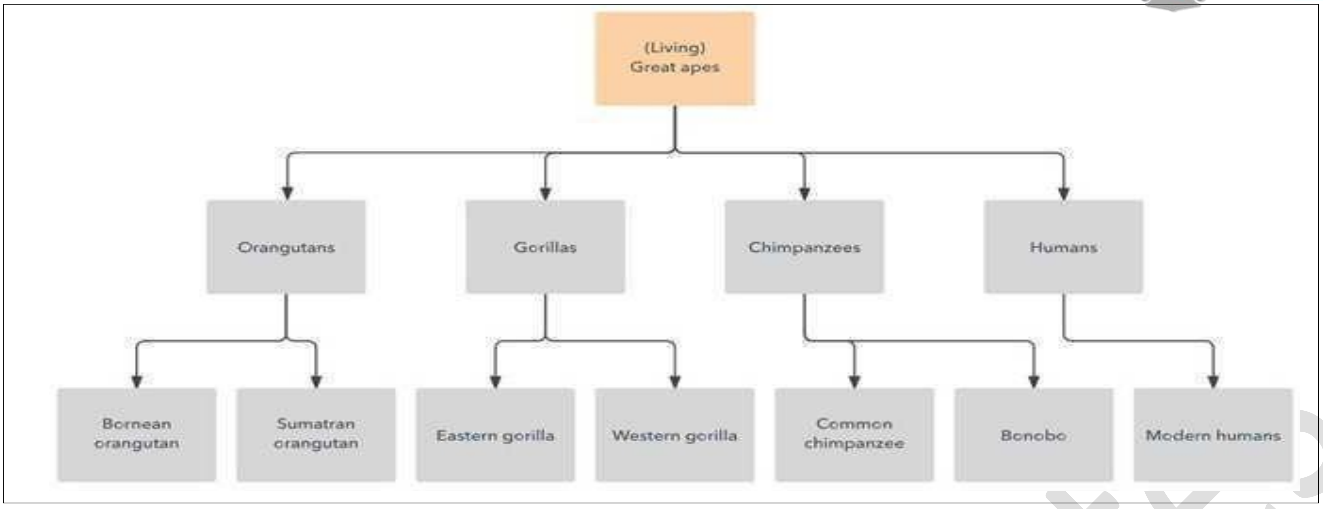
Within the database, tables can be normalised, or brought to comply with normalisation rules that make the database flexible, adaptable, and scalable. When normalised, each piece of data is atomic or broken into the smallest useful pieces.

Relational databases are typically written in Structured Query Language (SQL). E.F. Codd introduced the model in 1970.

The main highlights of this model are −

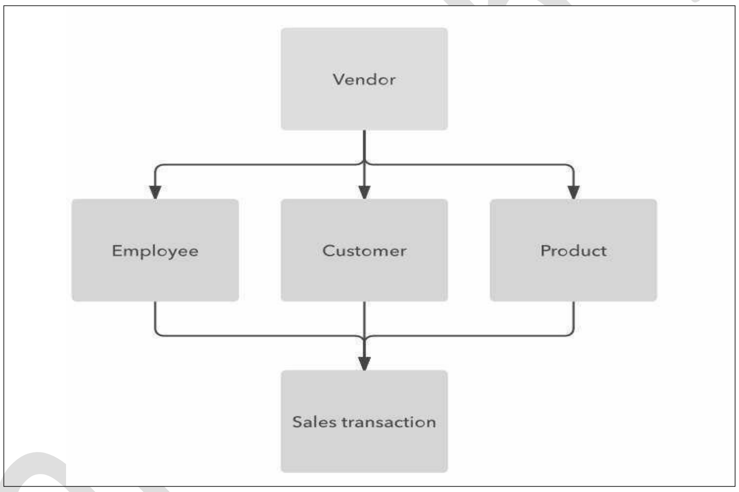
- Data is stored in tables called relations.
- Relations can be normalised.
- In normalised relations, values saved are atomic values.
- Each row in a relation contains a unique value.
- Each column in a relation contains values from the same domain.

Hierarchical model
The hierarchical model organises data into a tree-like structure, where each record has a single parent or root. Sibling records are sorted in an order. That order is used as the physical order for storing the database. This model is useful for describing many real-world relationships.



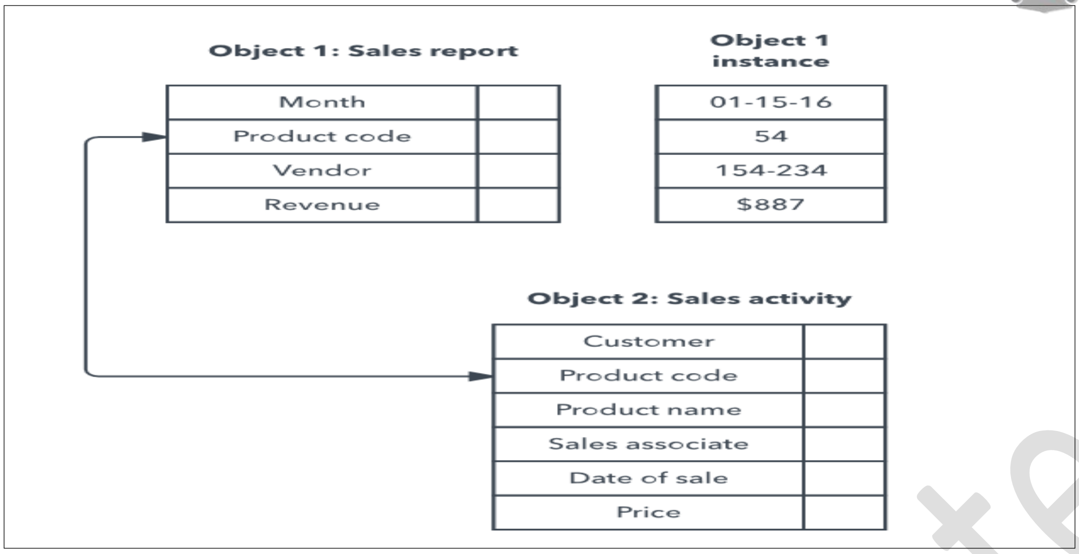
Network model

The network model builds on the hierarchical model by allowing many-to-many relationships between linked records, implying multiple parent records. Based on mathematical set theory, the model is constructed with sets of related records. Each set consists of one owner or parent record and one or more member or child records. A record can be a member or child in multiple sets, allowing this model to convey complex relationships.



Object-oriented database model

This model defines a database as a collection of objects, or reusable software elements, with associated features and methods. There are several kinds of object-oriented databases:
A multimedia database incorporates media, such as images, that could not be stored in a relational database.
A hypertext database allows any object to link to any other object. It is useful for organising lots of disparate data, but it is not ideal for numerical analysis.
The object-oriented database model is the best known post-relational database model, since it incorporates tables, but isn't limited to tables. Such models are also known as hybrid database models.



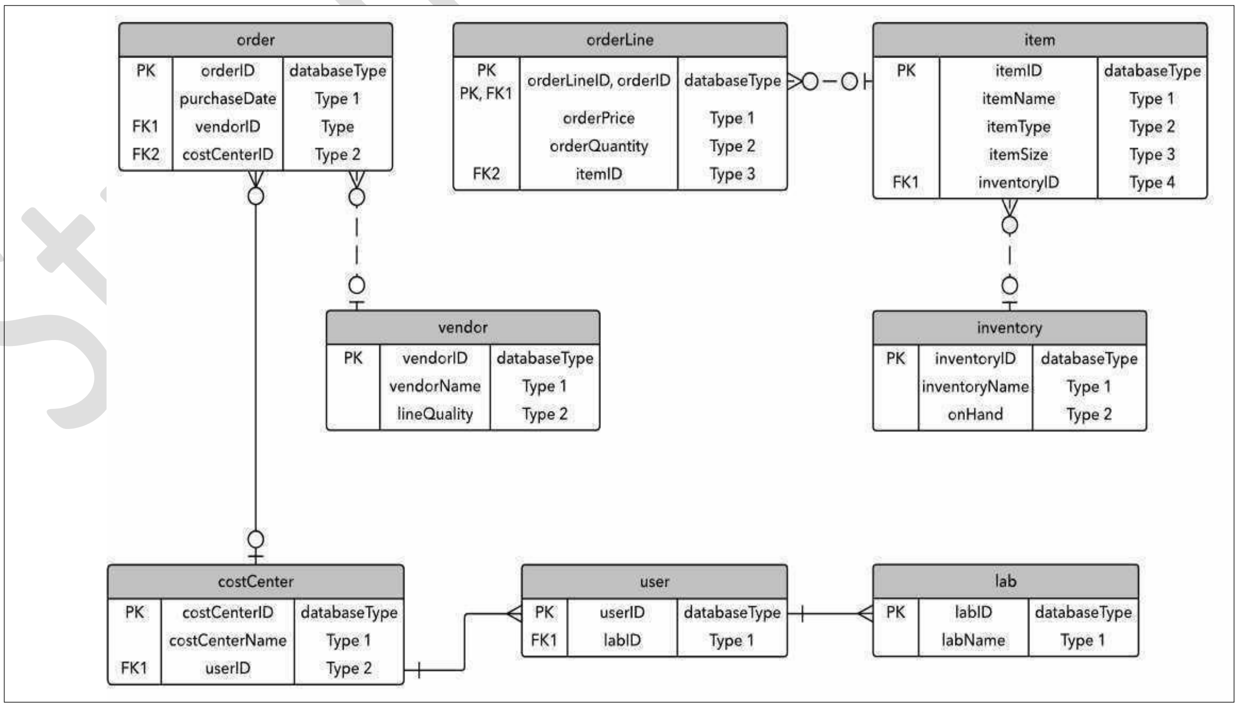
Object-relational model

This hybrid database model combines the simplicity of the relational model with some of the advanced functionality of the object-oriented database model. It allows designers to incorporate objects into the simple table structure.
Languages and call interfaces include SQL3, vendor languages, ODBC, JDBC, and proprietary call interfaces that are extensions of the languages and interfaces used by the relational model.

Entity-relationship model

This model captures the relationships between real-world entities much like the network model, but it is not as directly tied to the physical structure of the database. Instead, it is often used for designing a database conceptually.

Here, the people, places, and things about which data points are stored are referred to as entities, each of which has specific attributes that together make up their domain. The cardinality, or relationships between entities, are mapped as well.



NoSQL database models

In addition to the object database model, other non-SQL models have emerged in contrast to the relational model: The graph database model, which is even more flexible than a network model, allowing any node to connect with any other — the multi-valued model, which breaks from the relational model by allowing attributes to contain a list of data rather than a single data point.

Schema and Instance in DBMS
Design of a database is called the schema. The schema is of three types:

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

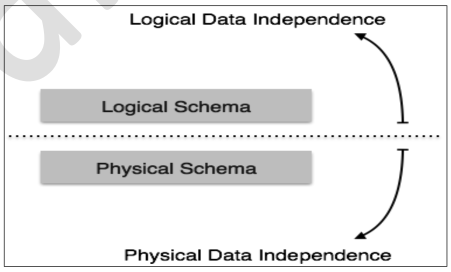
- **Logical schema: -** Design of database at a logical level is called logical schema, programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation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

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

Instances: - The data stored in a database at a moment of time is called an instance of the database. The database schema defines the variable declarations in tables that belong to a database the value of these variables at a moment of time is called the instance of that database.

Data Independence
A database system contains typically many data in addition to users' data. For example, it stores data about data, known as metadata, to locate and retrieve data quickly. It is rather difficult to modify or update a set of metadata once it is stored in the database. However, as a DBMS expands, it needs to change over time to satisfy the requirements of the users.

Logical Data Independence
Logical data is data about the 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 to that relation.

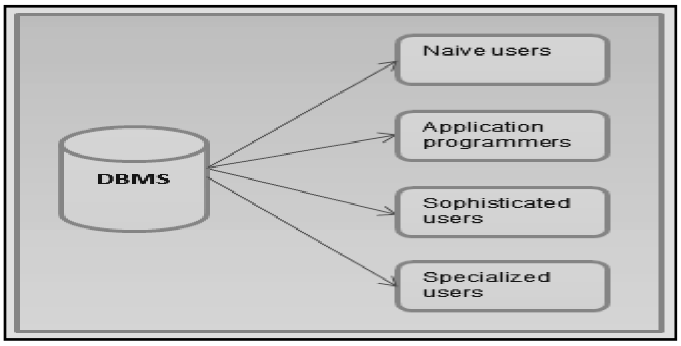


Logical data independence is a kind of mechanism, which liberalises itself from actual data stored on the disk. If we do some changes in table format, it should not change the data residing on the disk.

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. Requirements of the users. If the entire data is dependent, it will become a tedious and highly complex job.

DBA

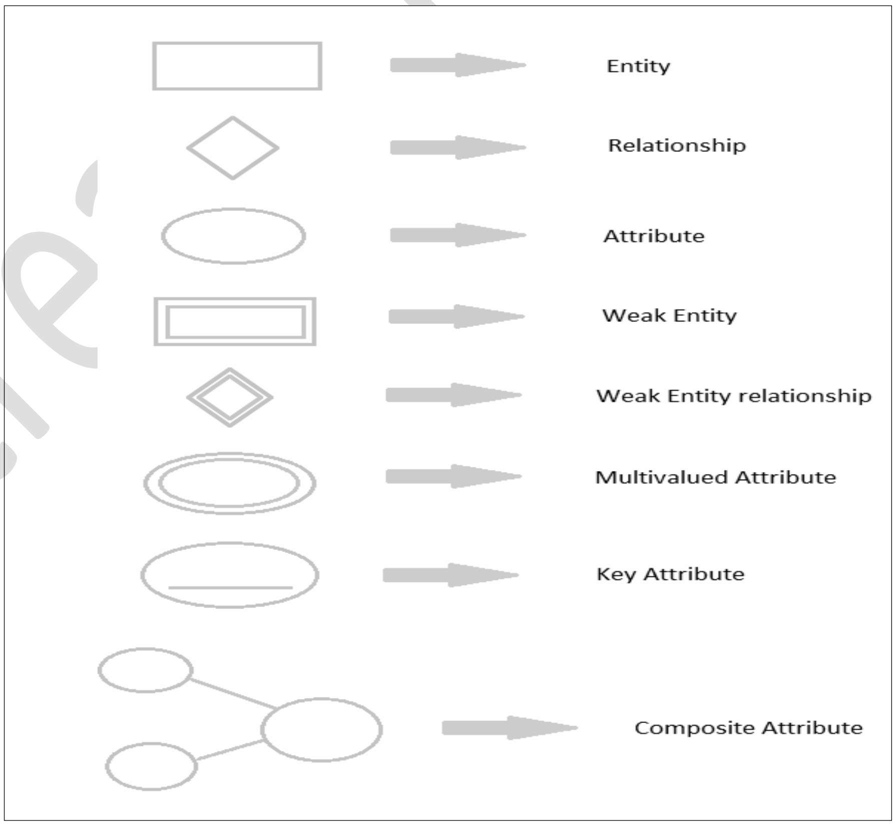


A data administration (also known as a database administration manager, data architect, or information centre manager) is a high-level function responsible for the overall management of data resources in an organisation. In order to perform its duties, the DA must know a good deal of system analysis and programming.

These are the functions of a data administrator (not to be confused with database administrator functions):

1. Data policies, procedures, standards
2. Planning- development of an organization's IT strategy, enterprise model, cost/benefit model, the design of the database environment, and administration plan.
3. Data conflict (ownership) resolution
4. Data analysis- Define and model data requirements, business rules, operational requirements, and maintain corporate data dictionary
5. Internal marketing of DA concepts
6. Managing the data repository

E-R Diagram



ER Model is represented using an ER diagram. Any object, for example, entities, attributes of an entity, relationship sets, and attributes of relationship sets, can be represented with the help of an ER diagram.

Entity
Entities are represented using rectangles. Rectangles are named with the entity set they represent.
EX. Student, Faculty, Course

- **Strong Entity: -** This type of entities has key attributes set that are used to create a primary key. Like student entity has Roll no as a critical attribute.
- **Weak Entity: -** This type of Entities does not have any primary key attribute, and they are dependent on some other entity have the crucial prime attribute. For example, Employee child is a weak entity some other entity has the crucial prime attribute. For example, Employee child is a weak entity which is dependent on the Employee Entity.
- **Composite Entity:** - This type of entities is used to replace many to many relationships, and Entity replaces that relationship. It is of three types one-to-one cardinality, one-to-many cardinality, many-to-many cardinality.

Attributes
Attributes are the properties of entities. Attributes are represented using ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).
Derived attributes: - This is the attributes that can be derived from other attributes. Like Age can be derived from the present date and Date of Birth.
Composite Attribute: - This is the attributes that can be created with the combination of two other attributes. Like a combination of first name and Last name will give the full name.
Single-value attribute: - This is the attributes which have single value when they are converted in the form of a table. Like DOB of a person.
Multi-Valued Attribute: - This is the attributes which have multiple values for the instance When an Entity is converted to a table, and that attribute allows multiple values for it. Like a person have two contact numbers.

3) Relationship
A Relationship describes relations between **entities**. The relationship is represented using diamonds.

There are three types of relationship that exist between Entities.

- Binary Relationship
- Recursive Relationship
- Ternary Relationship

Binary Relationship
Binary Relationship means the relation between two Entities. This is further divided into three types.

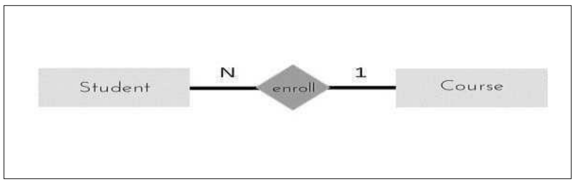
- **One to One:** This type of relationship is rarely seen in the real world.

The above example describes that one student can enrol only for one course and a course will also have only one Student. This is not what you will usually see in the relationship.

- **One to Many:** It reflects business rule that one entity is associated with any numbers of the same entity.

The example for this relation might sound a little weird, but this means that one student can enrol too many courses, but one course will have one Student.
The arrows in the diagram describe that one student can enrol for only one course.

- **Many to One:** It reflects a business rule that many entities can be associated with just one entity. For example, Student enrols for only one Course, but a Course can have many Students.

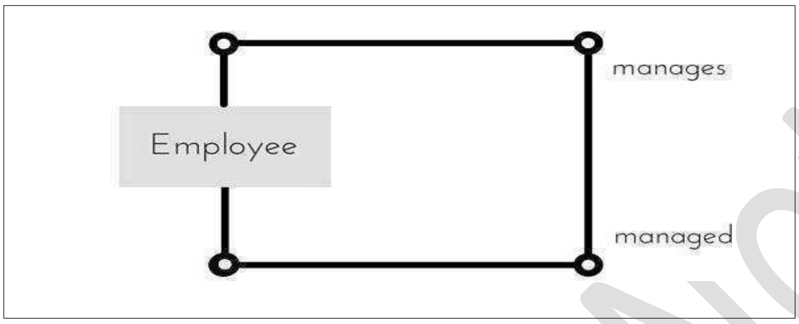


- **Many to Many:**

The above diagram represents that many students can enrol for more than one courses.

Recursive Relationship

When an Entity is related with itself, it is known as **Recursive** Relationship.



Ternary Relationship

Relationship of degree three is called Ternary relationship.

Here we are going to design an Entity Relationship (ER) model for a college database.
We have the following statements.

- A college contains many departments
- Each department can offer any number of courses
- Many instructors can work in a department
- An instructor can work only in one department
- For each department, there is a Head
- An instructor can be head of only one department
- Each instructor can take any number of courses
- Only one instructor can take a course
- A student can enrol for any number of courses
- Each course can have any number of students

Step 1: Identify the Entities

What are the entities here?
From the statements given, the entities are
Department
Course
Instructor
Student

Step 2: Identify the relationships

- One department offers many courses. However, one course can be offered by only one department. hence the cardinality between department and course is One to Many (1: N)
- One department has multiple instructors. However, instructor belongs to only one department. Hence the cardinality between department and instructor is One to Many (1: N)
- One department has only one head, and one head can be the head of only one department. Hence the cardinality is one to one. (1:1)
- One course can be enrolled by many students, and one student can enrol in many courses. Hence the cardinality between the course and the student is Many to Many (M: N)

- Only one instructor teaches one course. However, one instructor teaches many courses. Hence the cardinality between the course and the instructor is Many to One (N: 1)

Step 3: Identify the key attributes

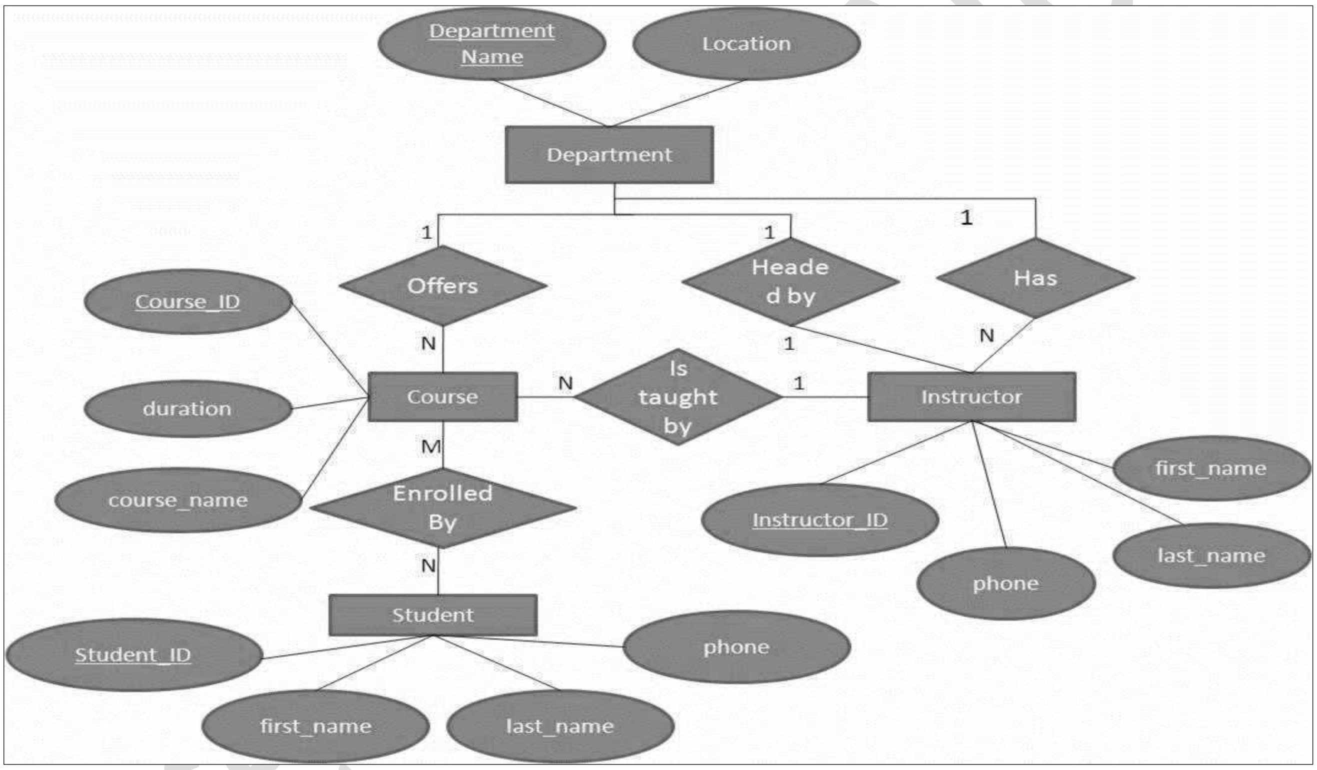
- "Departmen_Name" can identify a department uniquely. Hence Department Name is the key attribute for the Entity "Department".
- Course_ID is the critical attribute for "Course" Entity.
- Student_ID is the critical attribute for "Student" Entity.
- Instructor_ID is the critical attribute for "Instructor" Entity.

Step 4: Identify other relevant attributes

- For the department entity, other attributes are the location
- For course entity, other attributes are course_name, duration
- For instructor entity, other attributes are first_name, last name, phone
- For student entity, first_name, last_name, phone

Step 5: Draw a complete ER diagram

By connecting all these details, we can now draw an ER diagram as given below.



UNIT-II

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either unary or binary. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation, and intermediate results are also considered relations.

In the formal relational model terminology, a **row is called a tuple, a column header is called an attribute, and the table is called a relation.** A domain of possible values represents the data type describing the types of values that can appear in each column. We now define these terms—domain, tuple, attribute, and relation formally.

Domains, Attributes, Tuples, and Relations

Deposit Relation

bname	**Account**	**Ename**	**Balance**
Bhanwarkuwan	SBI1200	Ram	5000
Tilak Nagar	SBI1238	Amit	1000

Customer Relation

Ename	Street	City
Ramesh	MG road	Indore
Jhon	RNT Marg	Indore

- It has four attributes.
- For each attribute, there is a permitted set of values, called the **domain** of that attribute.
- E.g. the domain of *name* is the set of all branch names.

Let/denote the domain of *name*, and D_2, D_3 and D_4 the remaining attributes' domains respectively.

A domain D is a set of atomic values. By atomic we mean that each value in the domain is indivisible as far as the formal relational model is concerned. A common method of specifying a domain is to specify a data type from which the data values are forming the domain are drawn. It is also useful to specify a name for the domain, to help in interpreting its values. Some examples of domains follow:

- Usa_phone_numbers. The set of ten-digit phone numbers valid in the United States.
- Local_phone_numbers. The set of seven-digit phone numbers valid within an area code in the United States. The use of local phone numbers is quickly becoming obsolete, being replaced by standard ten-digit numbers.

A relation schema R, denoted by R (A 1, A 2, A n), is made up of a relation name R and a list of attributes, A 1, A 2,..., A n. Each attribute A i is the name of a role played by some domain D in the relation schema R. D is called the domain of A i and is denoted by dom (A i). A relation schema is used to describe a relation R is called the name of this relation. The degree (or arity) of a relation is the number of attributes n of its relation schema.

Using the data type of each attribute, the definition is sometimes written as:

STUDENT (Name: string, Ssn: string, Homophone: string, Address: string, Office phone: string, Age: integer, Gpa: real)

Characteristics of Relations

The Ordering of Tuples in a Relation. A relation is defined as a set of tuples. Mathematically, elements of a set have no order among them hence, tuples in a relation do not have any order.

The ordering of Values within a Tuple and an Alternative Definition of a Relation. According to the other definition of a relation, an n-tuple is an ordered list of n values, so the ordering of values in a tuple and hence of attributes in a relation schema is essential.

Values and NULLs in the Tuples. Each value in a tuple is an atomic value that is; it is not divisible into components within the framework of the underlying relational model. Hence, composite and multivalued attributes are not allowed. An important concept is that of NULL values, which are used to represent the values of attributes that may be unknown or may not apply to a tuple.

Relational Model Notation

A relation schema R of degree n is denoted by R (A 1, A 2, A n).

- The uppercase letters Q, R, S denote relation names.
- The lowercase letters q, r, s denotes relation states.
- The letters t, u, v denotes tuples.
- In general, the name of a relation schema such as STUDENT also indicates the current set of tuples in that relation—the current relation state—whereas STUDENT (Name, Ssn,) refers only to the relational schema.
- An attribute A can be qualified with the relation name R to which it belongs by using the dot notation R.A—for example, STUDENT. Name or STUDENT. Age. This is because the same name may be used for two attributes in different relations.

Relational Model Constraints and Relational Database Schemas

Constraints on databases can generally be divided into three main categories:
1. Constraints that are inherent in the data model. We call these inherent model-based constraints or implicit constraints. Example: In the relational model, no two tuples in a relation can be duplicates. Why? Because a relation is a set of tuples, as opposed to a bag/multiset or a sequence.
2. Constraints that can be directly expressed in schemas of the data model, typically by specifying them in the DDL. We call these schema-based constraints or specific constraints.
3. Constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs. We call these application-based or semantic constraints or business rules.

Types of Schema-based Constraints

Domain Constraints
Domain constraints specify that within each tuple, the value of each attribute A must be an atomic value from the domain dom(A).

Key Constraints and Constraints on NULL Values
In the formal relational model, a relation is defined as a set of tuples. All elements of a set are distinct; hence, all tuples in a relation must also be distinct.

A superkey SK specifies a uniqueness constraint that no two distinct tuples in any state r of R can have the same value for SK. Every relation has at least one default super key—the set of all its attributes. A superkey can have redundant attributes, however, so a more useful concept is that of a key, which has no redundancy.

A relation schema may have more than one key. In this case, each of the keys is called a **candidate key**. For example, the CAR relation has (**Licence_no Eng_sr_no** Model **Make_year** model) two candidate keys: License_number and Engine_serial_number. It is common to designate one of the candidate keys as the **primary key** of the relation. This is the candidate key whose values are used to identify tuples in the relation.

Relational Databases and Relational Database Schemas

The definitions and constraints we have discussed so far apply to single relations and their attributes. A relational database usually contains many relations, with tuples in relations that are related in many ways. In this section, we define a relational database and a relational database schema.
A relational database schema S is a set of relation schemas S = {R 1, R 2, ..., R m} and a set of integrity constraints IC. A relational database state 10 DB of S is a set of relation states DB = {r 1, r 2, ..., r m} such that each r i is a state of R i and such that the r i relation states satisfy the integrity constraints specified in IC. Below Figure shows a relational database schema that we call COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}. The underlined attributes represent primary keys.

EMPLOYEEE

Fname	Minit	Lname	Ssn	Bdata	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

DNAME	Dnumber	Mgr_ssn	Mgr_str_Date

DEPT_ LOCATIONS

Dnumber	DLocation

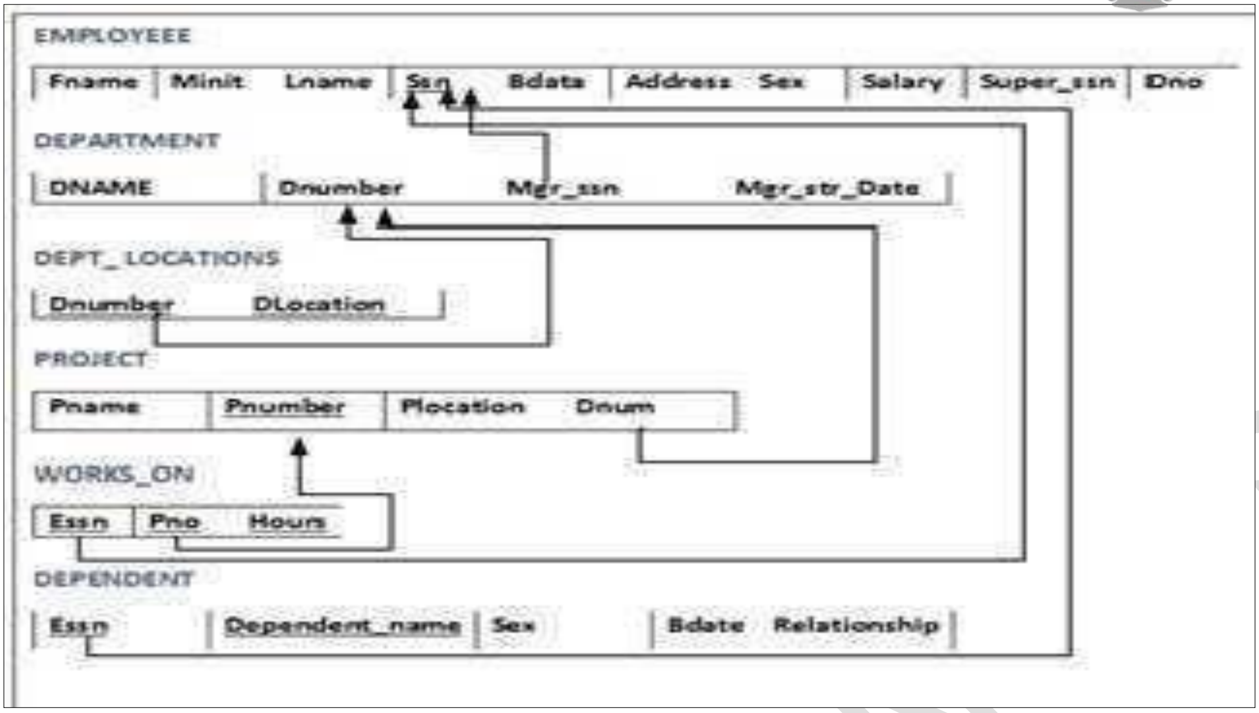
PROJECT

Pname	Pnumber	Plocation	Dnum

Integrity, Referential Integrity, and Foreign Keys

The entity integrity constraint states that no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples. For example, if two or more tuples had NULL for their primary keys, we may not be able to distinguish them if we try to reference them from other relations.

The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.



The relational algebra is essential for several reasons. First, it provides a formal foundation for relational model operations. Second, and perhaps more important, it is used as a basis for implementing and optimising queries in the query processing and optimisation modules that are integral parts of relational database management systems (RDBMSs).

Difference Between Calculus & algebra

BASIS FOR COMPARISON	**RELATIONAL ALGEBRA**	**RELATIONAL CALCULUS**
Basic	Relational Algebra is a Procedural language.	Relational Calculus is Declarative language.
States	Relational Algebra states how to obtain the result.	Relational Calculus states what result we must obtain.
Order	Relational Algebra describes the order in which operations must be performed.	Relational Calculus does not specify the order of operations.
Domain	Relational Algebra is not domain dependent.	Relation Calculus can be domain dependent.
Related	It is close to a programming language.	It is close to the natural language.

The fundamental operations of relational algebra are as follows −

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

We will discuss all these operations in the following sections.

Select Operation (σ)
It selects tuples that satisfy the given predicate from a relation.

Notation – σ(r)
Where σ stands for selection predicate and r stands for relation. p is the propositional logic formula which may use connectors like and, or, and not. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example −
σsubject = "database"(Books)
Output − Selects tuples from books where the subject is 'database'.
σsubject = "database" and price = "450"(Books)
Output − Selects tuples from books where the subject is 'database', and 'price' is 450.
σsubject = "database" and price = "450" or year > "2010"(Books)
Output − Selects tuples from books where the subject is 'database', and 'price' is 450 or those books published after 2010.

Project Operation (∏)
It projects column(s) that satisfy a given predicate.

Notation − ∏A1, A2, An (r)

Where A1, A2, An are attribute names of relation r.
Duplicate rows are automatically eliminated, a the relation is a set.

For example −
∏the subject, author (Books)
Selects and projects columns named as subject and author from the relation Books.

Union Operation (U)
It performs a binary union between two given relations and is defined as −
r U s = { t | t ∈ r or t ∈ s}
Notation − r U s

Where r and s are either database relations or relation result set (temporary relation).
For a union operation to be valid, the following conditions must hold −

r and s must have the same number of attributes.
Attribute domains must be compatible.
Duplicate tuples are automatically eliminated.
∏ author (Books) U ∏ author (Articles)
Output − Projects the names of the authors who have either written a book or an article or both.

Set Difference (−)
The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation − r − s
Finds all the tuples that are present in r but not in s.

∏ author (Books) − ∏ author (Articles)
Output − Provides the name of authors who have written books but not articles.

Cartesian Product (X)
Combines information of two different relations into one.
Notation – r X s
Where r and s are relations and their output will be defined as –

r X s = { q t | q ∈ r and t ∈ s}
σauthor = 'tt'(Books X Articles)
Output – Yields a relation, which shows all the books and articles written by tt.

Rename Operation (p)
The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter rho p.

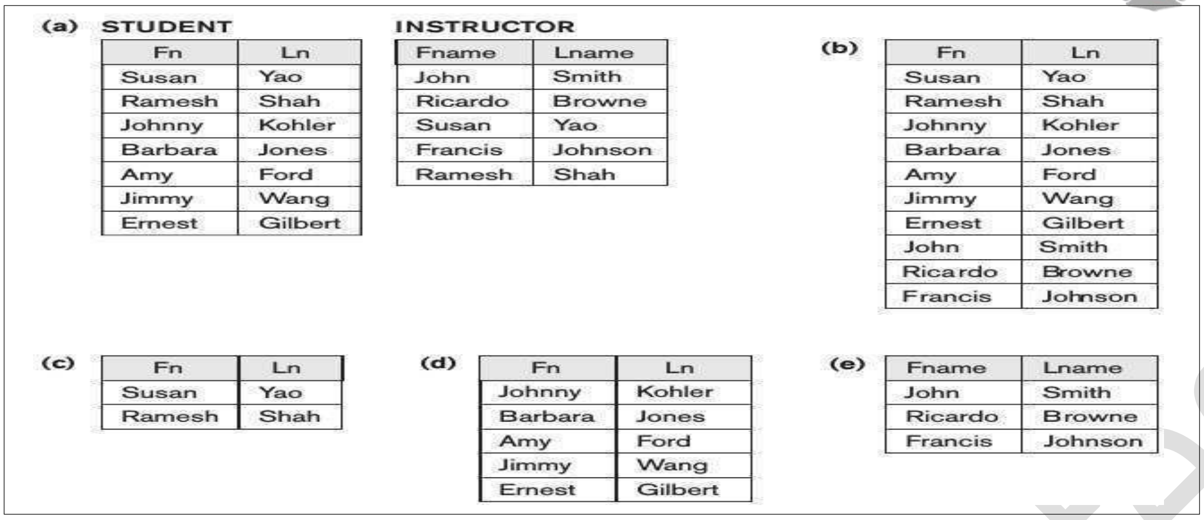
Notation – p x (E)
Where the result of expression E is saved with the name of x.
Additional operations are –

- Set intersection
- Assignment
- Natural join

We can define the three operations UNION, INTERSECTION, and SET DIFFERENCE
on two union-compatible relations R and S as follows:
■ UNION: The result of this operation, denoted by R U S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.
■ INTERSECTION: The result of this operation, denoted by ' ∩ S, is a relation that includes all tuples that are in both R and S.
■ SET DIFFERENCE (or MINUS): The result of this operation, denoted by R – S, is a relation that includes all tuples that are in R but not in S.
The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations.
(b) STUDENT U INSTRUCTOR. (c) STUDENT ∩ INST'UCTO'. (d) STUDENT − INST'UCTO'.
(e) INST'UCTO' – STUDENT.

CARTESIAN PRODUCT operation—also known as CROSS PRODUCT or CROSS JOIN—which is denoted by **×.** This is also a binary set operation, but the relations on which it is applied do not have to be union-compatible. The JOIN operation, denoted by, is used to combine related tuples from two relations into single "longer" tuples. This operation is very important for any relational database with more than a single relation because it allows us to process relationships among relations.

DEPT_MG' $\leftarrow$ DEPA'TMENT $\bowtie_{\text{Mgr\_ssn=Snn}}$ EMPLOYEE
'ESULT $\leftarrow$ $\prod_{\text{Dame, Lname}}$ (DEPT_MGR)



(a) STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

(b)

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

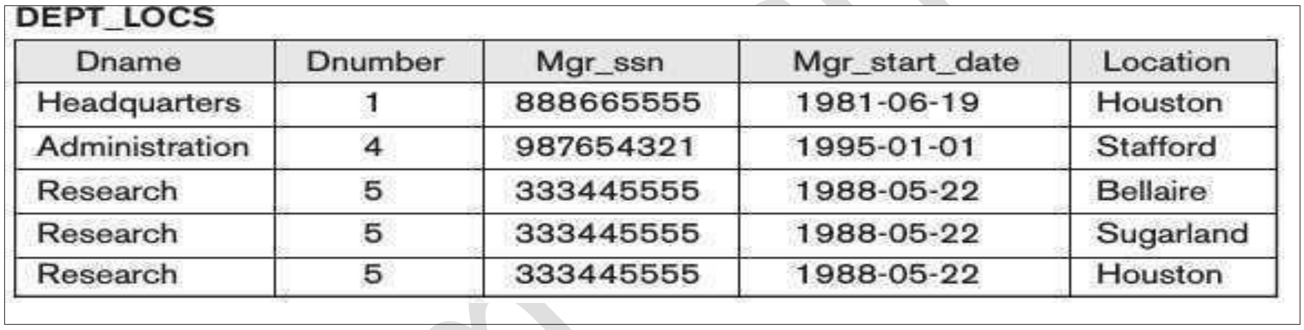
Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Variations of JOIN: The EQUIJOIN and NATURAL JOIN
The most common use of JOIN involves join conditions with equality comparisons only. Such a JOIN, where the only comparison operator used is =, is called an EQUIJOIN.

The standard definition of NATURAL JOIN requires that the two join attributes (or each pair of join attributes) have the same name in both relations. If this is not the case, a renaming operation is applied first.
P‘OJ_DEP DEPT_LOCS ← DEPA‘TMENT * DEPT_LOCATIONS.



DEPT_LOCS

Dname	Dnumber	Mgr_ssn	Mgr_start_date	Location
Headquarters	1	888665555	1981-06-19	Houston
Administration	4	987654321	1995-01-01	Stafford
Research	5	333445555	1988-05-22	Bellaire
Research	5	333445555	1988-05-22	Sugarland
Research	5	333445555	1988-05-22	Houston

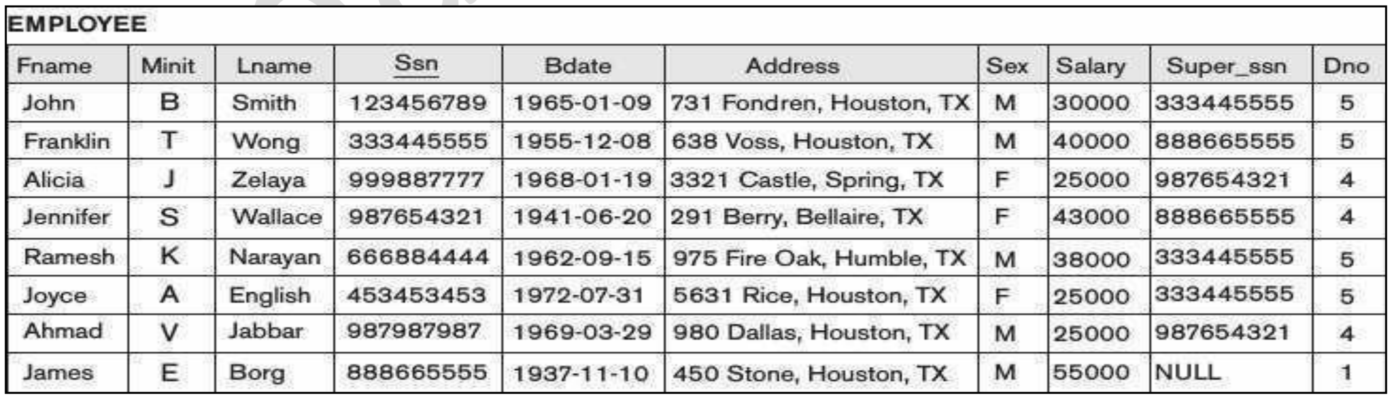
The DIVISION Operation
The DIVISION operation, denoted by ÷, is useful for a special kind of query that sometimes occurs in database applications. An example is ‘etrieve the names of employees who work on all the projects that ‘John Smith’ works on.

$$\textbf{SMITH} \leftarrow \sigma_{\text{Fname ='Jhon'}} \textbf{ AND Lname = }_{\text{'SMITH'}} \textbf{ (EMPLOYEE)}$$
$$\textbf{SMITH\_PNOS} \leftarrow u_{\text{pno}} \textbf{ (WORKS\_ON} \bowtie_{\text{Essn = Ssn}} \textbf{SMITH)}$$

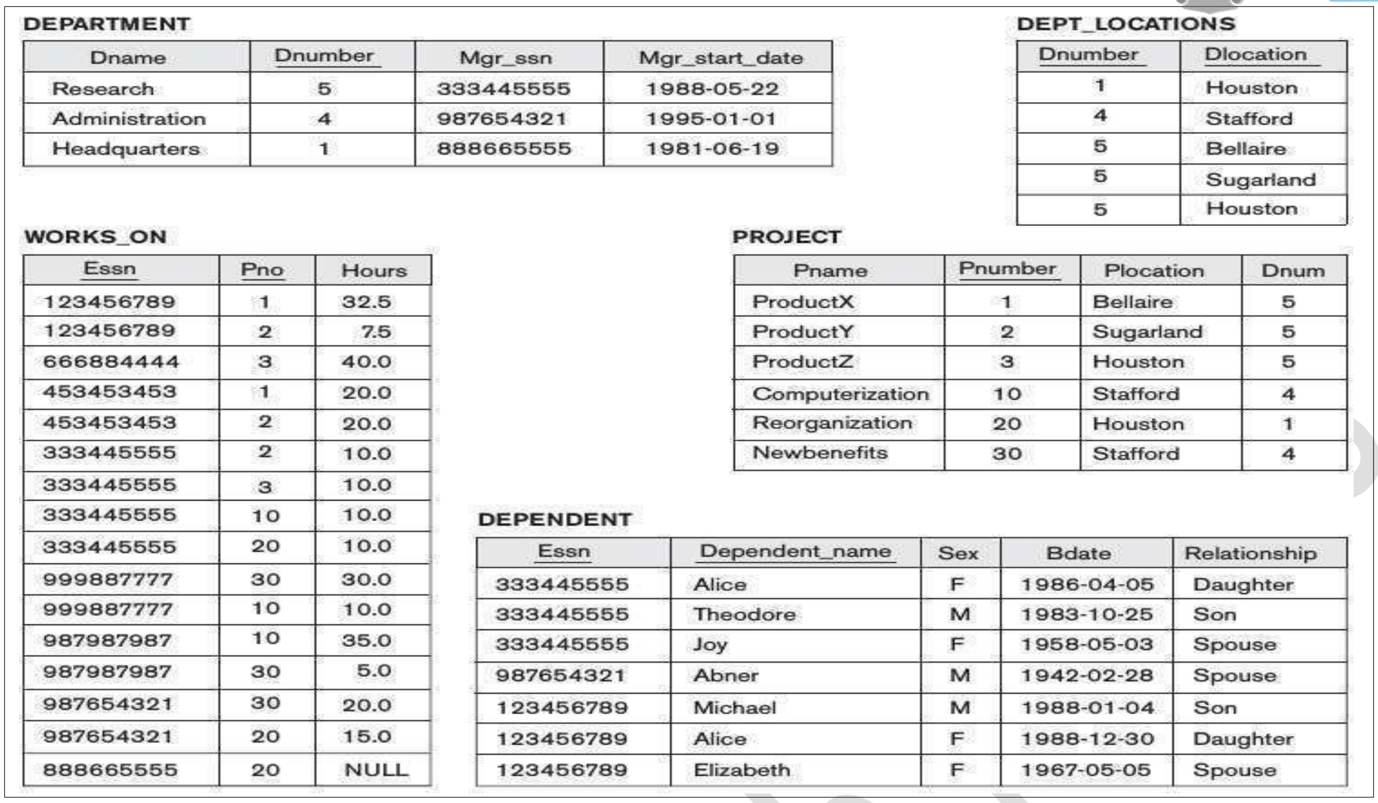
	PURPOSE	**NOTATION**
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	σ<selection condition>(R)
PROJECT	Produces a new relation with only some of the attributes of R and removes duplicate tuples.	u<attribute

THETA JOIN	Produces all combinations of tuples from R and R1 2 that satisfy the join condition.	R1 <join
EQUIJOIN	Produces all the combinations of tuples from R1 and R2 that satisfy a join condition with only equality comparisons.	R1 <join
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	R1*<join condition> R2, OR R1* (<join
UNION	Produces a relation that includes all the tuples in R1 or R2 or both R1 and R2; R1 and R2 must be union-compatible.	R1 U R2
INTERSECTION	Produces a relation that includes all the tuples in both R1 and R2; R1 and R2 must be union-compatible.	‘1 ∩ ‘2
DIFFERENCE	Produces a relation that includes all the tuples in R1 that are not in R2; R1 and R2 must be union-compatible.	R1 – R2
CARTESIAN PRODUCT	Produces a relation that has the attributes of R1 and R2 and includes as tuples all possible combinations of tuples from R1 and R2.	R1 × R2
DIVISION	Produces a relation R(X) that includes all tuples t[X] in R1(Z) that appear in R1 in combination with every tuple from R(Y), where Z = X U Y.	R1(Z) ÷ R2(Y)

Example on Relational Algebra


EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1



DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Query 1. Retrieve the name and address of all employees who work for the ‘‘esearch’ department.
‘ESEA‘CH_DEPT ← σ $_{\text{Dname = 'Research'}}$(DEPARTMENT)
‘ESEA‘CH_EMPS ← (‘ESEA‘CH_DEPT ⋈ $_{\text{Dnumber = Dno}}$ EMPLOYEE)
‘ESULT← u $_{\text{Fname, Lname, Address}}$ (RESEARCH_EMPS)

Query 2. For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, address, and birth date.

STAFFO‘D_P‘OJs ← $_{\sigma\ \text{Plocation = 'stafford'}}$(PROJECT)
CONTR_DEPTS← (STAFFORD_PROJs⋈ $_{\text{Dnum=Dnumber}}$ DEPARTMENT)
P‘OJ_DEPT_MG‘S ← (CONT‘_DEPTS ⋈ $_{\text{Mgr\_ssn = Ssn}}$ EMPLOYEE)
‘ESULT ← u $_{\text{Pnumber, Dnum, Lname, Address,Bdate}}$(PROJ_DEPT_MGRS)
Query 3. Find the names of employees who work on all the projects controlled by department number 5.

DEPT5_P‘OJS ← p(Pno)(uPnumber(σDnum=5(PROJECT)))
EMP_P‘OJ ← p (Ssn, Pno)(uEssn, Pno(WO‘KS_ON))
‘ESULT_EMP_SSNS ← EMP_P‘OJ ÷ DEPT5_P‘OJS
‘ESULT ← uLname, Fname (RESULT_EMP_SSNS * EMPLOYEE)

Query 4. List the names of all employees with two or more dependents.

Strictly speaking, this query cannot be done in the primary (original) relational algebra. We must use the AGGREGATE FUNCTION operation with the COUNT aggregate function. We assume that dependents of the same employee have distinct Dependent_name values.

T1(Ssn, No_of_Dependents) ← $_{\text{Essn}}$ ℑ COUNT $_{\text{Dependent\_name}}$(DEPENDENT)
T2 ← σ $_{\text{No\_of Dependent} \geq 2}$(T1)
‘ESULT ← $_{\text{u Lname, Fname}}$ (T2* EMPLOYEE)

ID	Name	Dept_name	Salary
10101	Shrinivasan	Comp.sci	65000

12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstin	Physics	95000

Instructor Relation
instructor relation where the instructor is in the "Physics" department, we write:
dept name = "Physics" (instructor)

Relational Calculus

In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.

'elational calculus exists in two forms –

Tuple Relational Calculus (TRC) Filtering variable ranges over tuples

Notation – {T | Condition}

Returns all tuples T that satisfies a condition.

For example –

{ T.name | Author(T) AND T.article = 'database' }
Output – 'eturns tuples with 'name' from Author who has written article on 'database'.

TRC can be quantified. We can use Existential (∃) and Universal Quantifiers (∀).

For example –

{ R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}
Output – The above query will yield the same result as the previous one.

Domain Relational Calculus (DRC)
In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

Notation –

{ a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

Where a1, a2 are attributes and P stands for formulae built by inner attributes.

For example –

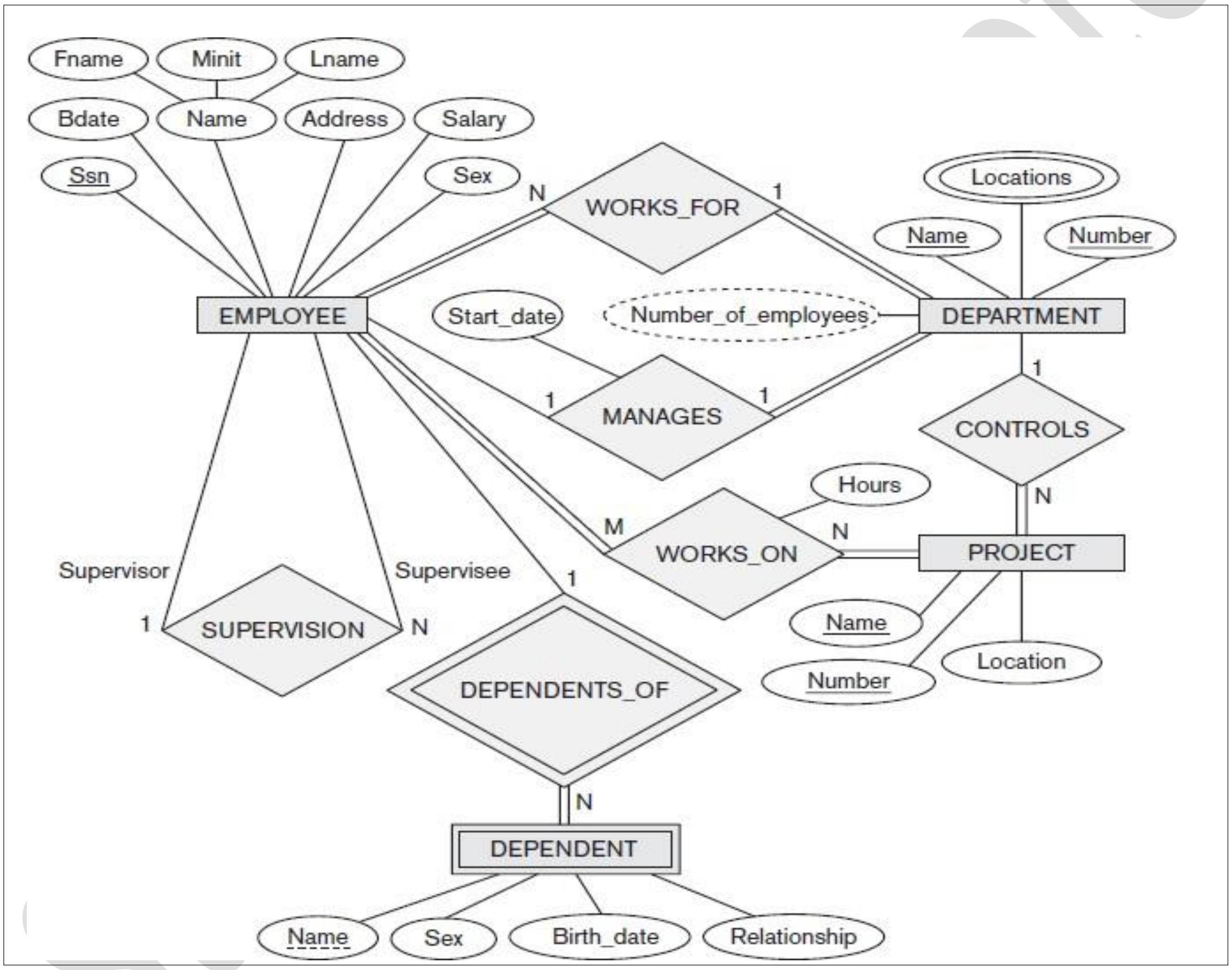
{< article, page, subject > | ∈ TP A subject = 'database'}
Output – Yields Article, Page, and Subject from the relation TP where subject is database.

Just like TRC, DRC can also be written using existential and universal quantifiers. DRC also involves relational operators.
The expression power of Tuple Relation Calculus and Domain Relational Calculus is equivalent to Relational

Algebra.

E-R Diagram
An ER schema diagram for the COMPANY database
Entities and Attributes
Entities and Their Attributes. The basic object that the ER model represents is an **entity**, which is a *thing* in the real world with an independent existence. An entity may be an object with a physical existence (for example, a person, car, house, or employee) or it may be an object with a conceptual existence (for instance, a company, a job, or a university course).

Composite versus Simple (Atomic) Attributes. Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings — for example, the Address attribute of the EMPLOYEE entity.



Single-Valued versus Multivalued Attributes. Most attributes have a single value for a particular entity; such attributes are called **single-valued**. For example, Age.
A multivalued attribute may have lower and upper bounds to constrain the *number of values* allowed for each entity. For example, the Colors attribute of a car.
Stored versus Derived Attributes. In some cases, two (or more) attribute values are related—for example, the Age and Birth_date attributes of a person.

Initial Conceptual Design of the COMPANY Database covers
Relationship & type
Cardinality

Week Entity

Participation Constraints: - The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type.

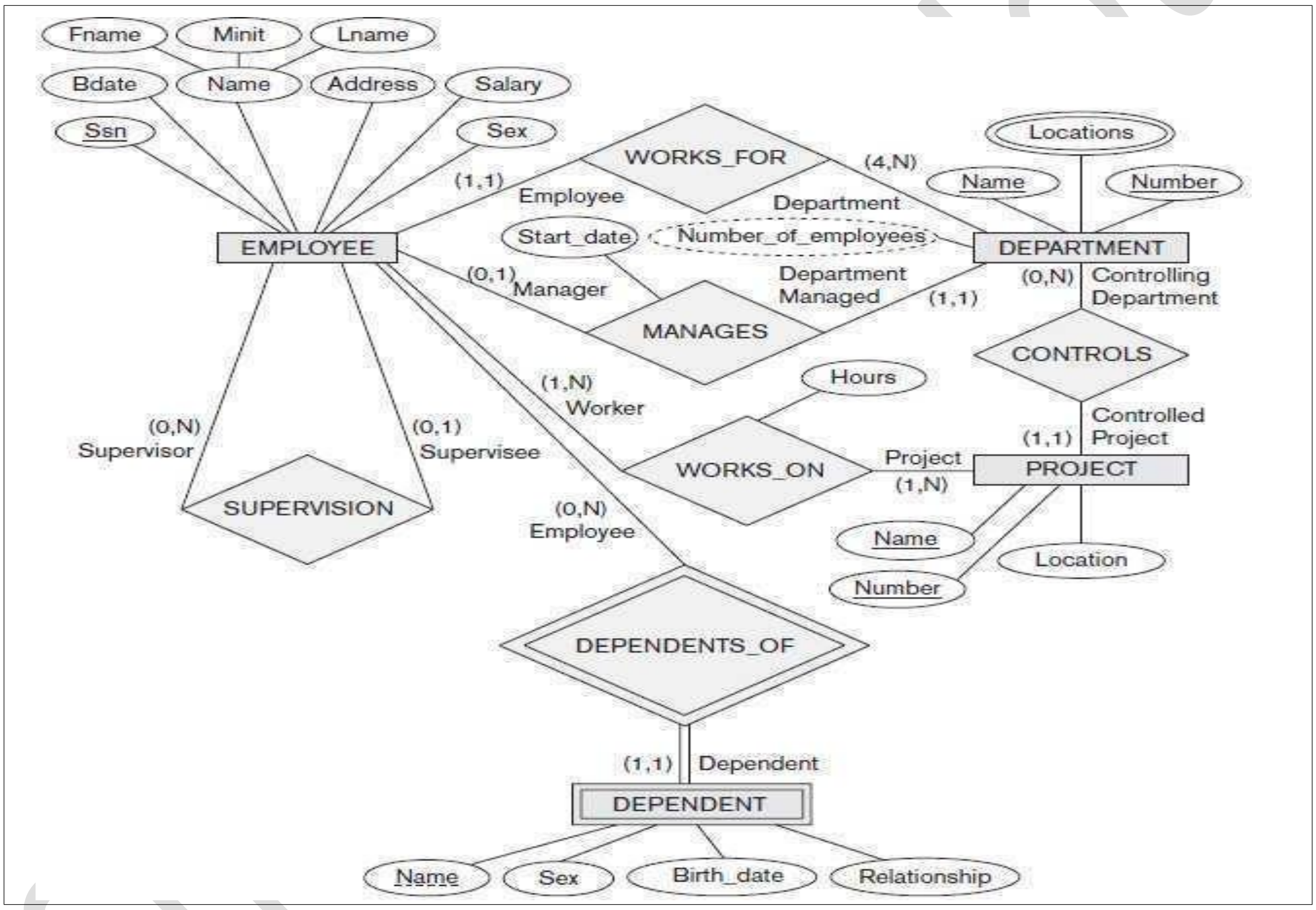
Company schema, with structural constraints specified using (min, max) notation and role names.

Enhanced Entity-Relationship (EER) Model

Semantic data modelling concepts that were incorporated into conceptual data models such as the ER Model. ER model can be enhanced to include these concepts, leading to the **Enhanced ER (EER**) model.

Subclasses: - An entity type is used to represent both a type of entity and the entity set or collection of entities of that type that exist in the database. For example, the entity type EMPLOYEE describes the type (that is, the attributes and relationships) of each employee entity, and refers to the current set of EMPLOYEE entities in the COMPANY database.

Superclasses: - We call each of these subgroupings a subclass or subtype of the EMPLOYEE entity type, and the EMPLOYEE entity type is called the superclass or supertype for each of these subclasses.



Extended E-R Model

Specialisation

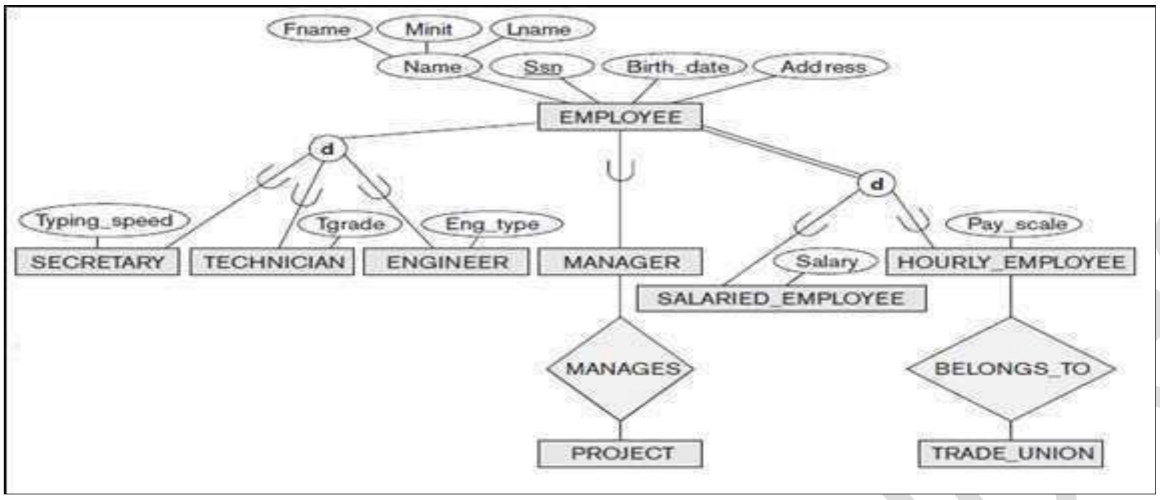
Specialisation is the process of defining a *set of subclasses* of an entity type this entity type is called the **superclass** of the specialisation. The set of subclasses that forms a specialisation is defined by some distinguishing characteristic of the entities in the superclass. For example, the set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialisation of the superclass EMPLOYEE that distinguishes among employee entities based on the *job type* of each employee entity.

Steps for Specialization

- Define a set of subclasses of an entity type.
- Establish additional specific attributes with each subclass.
- Establish additional specific relationship types between each subclass and other entity types or other subclasses.

EER diagram notation to represent subclasses and specialisation. Three specialisations of EMPLOYEE:

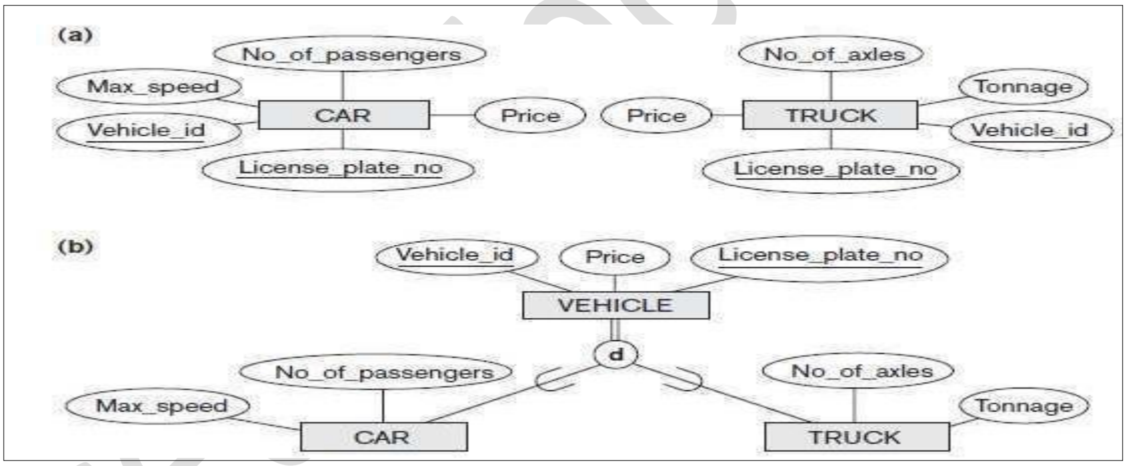
{SECRETARY, TECHNICIAN, ENGINEER}
{MANAGER}
{HOURLY_EMPLOYEE, SALARIED_EMPLOYEE}



Generalisation

Generalisation

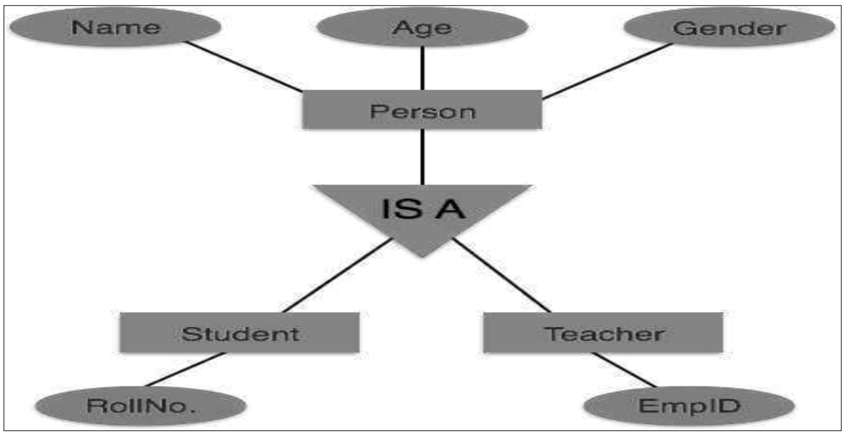
We can think of a *reverse process* of abstraction in which we suppress the differences among several entity types, identify their standard features, and **generalise** them into a single **superclass** of which the original entity types are subclasses. For example, consider the entity types CAR and TRUCK.
Generalisation. (a) Two entity types, CAR and TRUCK. (b)Generalising CAR and TRUCK into the superclass VEHICLE.



Specialization

Inheritance

We use all the above features of ER-Model to create classes of objects in object-oriented programming. The details of entities are generally hidden from the user; this process known as abstraction.
Inheritance is an essential feature of Generalization and Specialization. It allows lower-level entities to inherit the attributes of higher-level entities.

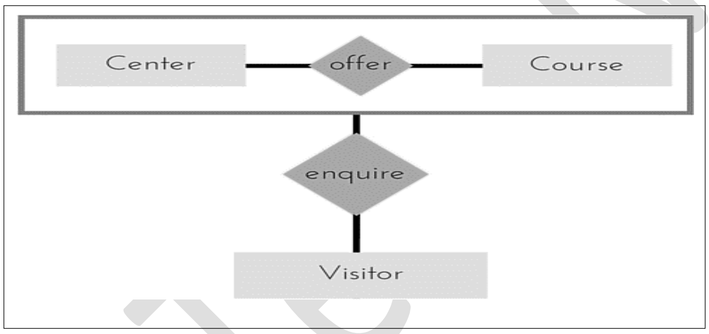


Inheritance

For example, the attributes of a Person class such as name, age, and gender can be inherited by lower-level entities such as Student or Teacher.

Aggregation

Aggregation is a process when the relation between two entities is treated as a single entity. Here the relation between Center and Course is acting as an Entity in relation with Visitor.



Aggregation

UNIT III

Introduction to SQL Structure Query Language (SQL) is a programming language used for storing and managing data in RDBMS. SQL was the first commercial language introduced for E.F Codd's Relational model. Today almost all RDBMS (MySQL, Oracle, Infomax, Sybase, MS Access) uses SQL as the standard database language. SQL is used to perform all type of data operations in RDBMS.

SQL Command

DDL: Data Definition Language

All DDL commands are auto-committed. That means it saves all the changes permanently in the database.

Command	Description
create	to create a new table or database
alter	for alteration
truncate	delete data from a table
drop	to drop a table
rename	to rename a table

DML: Data Manipulation Language

DML commands are not auto-committed. It means changes are not permanent to the database; they can be rolled back.

Command	Description
insert	to insert a new row
update	to update an existing row
delete	to delete a row
merge	merging two rows or two tables

TCL: Transaction Control Language

These commands are to keep a check on other commands and their effect on the database. These commands can annul changes made by other commands by rolling back to the original state. It can also make changes permanent.

Command	Description
commit	to permanently save
rollback	to undo the change
save point	to save temporarily

DCL: Data Control Language

Data control language provides a command to grant and take back authority.

Command	Description
grant	grant permission of the right

revoke	take back permission.

DQL: Data Query Language

Command	**Description**
select	retrieve records from one or more table

Create command
create is a DDL command used to create a table or a database.
Creating a Database
To create a database in RDBMS, *create* command is used. Following is the Syntax,
create database *database-name*;
Example for Creating Database
create database Test;
The above command will create a database named **Test**.
Creating a Table
Create command also used to create a table. We can specify names and datatypes of various columns along. Following is the Syntax,
create table *table-name*
{
 column-name1 datatype1,
 column-name2 datatype2,
 column-name3 datatype3,
 column-name4 datatype4
};
create table command will tell the database system to create a new table with the given table name and column information.
Example for creating Table
create table Student (id int, name varchar, age int);

alter command
Alter command used for alteration of table structures. There are various uses of *altering* command, such as,

- to add a column to the existing table
- to rename any existing column
- to change the datatype of any column or to modify its size.
- *Alter* is also used to drop a column.

Using alter command we can add a column to an existing table. Following is the Syntax,

alter table table-name add (column-name datatype);
Here is an Example for this, **alter table Student add (address char);**

To Add a column with Default Value
alter command can add a new column to an existing table with default values. Following is the Syntax,
alter table table-name add (column-name1 datatype1 default data);
Example **alter table Student add (dob date default '1-Jan-99');**

To Modify an Existing Column
Alter command used to modify data type of an existing column. Following is the Syntax,
alter table table-name modify (column-name datatype);
Here is an Example for this, **alter table Student modify (address varchar (30));**

The above command will modify the address column of the Student table
To Rename a column
Using alter command you can rename an existing column.
alter table table-name rename old-column-name to column-name;
Here is an Example for this, **alter table Student rename address to Location;**
The above command will rename address column to Location.
To Drop a Column
alter command also used to drop columns also.
alter table table-name drop(column-name);
Here is an Example for this, **alter table Student drop(address);**
The above command will drop the address column from the Student table.

truncate command
The truncate command removes all records from a table. However, this command will not destroy the table's structure. When we apply to truncate command on a table its Primary key is initialised. Following is its Syntax,
truncate table table-name
Example **truncate table Student;**

drop command
Drop query completely removes a table from the database. This command will also destroy the table structure. Following is its Syntax,
drop table table-name
Here is an Example explaining it. **Drop table Student;**

rename query
Rename command is used to rename a table. Following is its Syntax,
rename table old-table-name to new-table-name
Here is an Example explaining it. **Rename table Student to Student-record;**

DML Commands
1) INSERT command
Insert command is used to insert data into a table. Following is its general syntax,
INSERT into table-name values (data1, data2,)
example,
Consider a table Student with following fields.

S_id	S_Name	age

INSERT into Student values(101,'Adam',15);
The above command will insert a record into Student table.

S_id	S_Name	age
101	Adam	15

Example to Insert NULL value to a column
Both the statements below will insert a NULL value into the age column of the Student table.
INSERT into Student (id, name) values(102,'Alex');
Alternatively,
INSE‘T into Student values (102,'Alex’, null);
The above command will insert only two column value another column is set to null.

S_id	S_Name	age
101	Adam	15
102	Alex	

Example to Insert Default value to a column
INSE‘T into Student values (103,'Chris’, default)

S_id	S_Name	age
101	Adam	15
102	Alex	
103	Chris	14

Suppose the age column of student table has a default value of 14.

2) UPDATE command
Update command is used to update a row of a table. Following is its general syntax,
UPDATE table-name set column-name = value where condition;
example,
update Student set age=18 where s_id=102;

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	chris	14

Example
UPDATE Student set s_name='Abhi’, age=17 where s_id=103;
The above command will update two columns of a record.

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

3) Delete command
Delete command is used to delete data from a table. Delete command can also be used with the condition to delete a row. Following is its general syntax,

DELETE from table-name;
Example
DELETE from Student;
The above command will delete all the records from the Student table.
Example to Delete a Record from a Table
Consider the following Student table

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

DELETE from Student where s_id=103;
The above command will delete the record where s_id is 103 from the Student table.

S_id	S_Name	age
101	Adam	15
102	Alex	18

TCL command
Transaction Control Language (TCL) commands are used to manage transactions in the database. These are used to manage the changes made by DML statements. It also allows statements to be grouped into logical transactions.
Commit command
Commit command is used to save any transaction into the database permanently.

Following is Commit command's syntax,
commit;

Rollback command

This command restores the database to the last committed state. It is also used with savepoint command to jump to a save point in a transaction.

Following is Rollback command's syntax,
rollback to savepoint-name;
Save point command
savepoint command used to temporarily save a transaction so that you can roll back to that point whenever necessary.
Following is save point command's syntax,
savepoint savepoint-name;

DCL command

System: creating a session, table etc. are all types of system privilege.
Object: any command or query to work on tables comes under object privilege.
DCL defines two commands,
Grant: Gives user access privileges to the database.
Revoke: Take back permissions from the user.
Example **grant create session to username;**

S_id	**s_Name**	**age**	**address**
101	Adam	15	Noida
102	Alex	18	Delhi
103	Abhi	17	Rohtak
104	Ankit	22	Panipat

WHERE clause

Where clause is used to specify condition while retrieving data from the table. Where clause is used mostly with Select, Update and Delete query. If the condition specified by where clause is true, then only the result from the table is returned.

The syntax for WHERE clause

```
SELECT column-name1,
 column-name2,
 column-name3,
 column-name N
from table-name WHERE [condition];
```

Example SELECT s_id, s_name, age, address
from Student WHERE s_id=101;

SELECT Query

The Select query is used to retrieve data from a table. It is the most used SQL query. We can retrieve complete tables, or partial by mentioning conditions using WHERE clause.

Syntax of SELECT Query
SELECT column-name1, column-name2, column-name3, column-nameN from table-name;
Example **SELECT s_id, s_name, age from Student.**

Like clause
Like clause is used as a condition in SQL query. Like clause compares data with an expression using wildcard operators. It is used to find similar data from the table.

Wildcard operators
Two wildcard operators are used in like clause.
Per cent sign % represents zero, one or more than one character.
Underscore sign _: represents only one character.
Example: - SELECT * from Student where s_name like 'A%';

Order by Clause
Order by clause is used with a Select statement for arranging retrieved data in sorted order. The Order by clause by default sort data in ascending order. To sort data in descending order DESC keyword is used with Order by clause.

The syntax of Order By
SELECT column-list|* from table-name order by asc|desc;
Example SELECT * from Emp order by salary;

Group by Clause
Group by clause is used to group the results of a SELECT query based on one or more columns. It is also used with SQL functions to group the result from one or more tables.
The syntax for using Group by in a statement.

SELECT column_name, function(column_name)
FROM table_name
WHERE condition
GROUP BY column_name
Example select name, salary
from Emp
where age > 25
group by salary

HAVING Clause
Having clause is used with SQL Queries to give a more precise condition for a statement. It is used to mention condition in Group based SQL functions, just like WHERE clause.
Syntax for having will be,
select column_name, function(column_name)
FROM table_name
WHERE column_name condition
GROUP BY column_name
HAVING function(column_name) condition
Example SELECT *
from sale group customer
having sum(previous_balance) > 3000

Distinct keyword

The distinct keyword is used with a Select statement to retrieve unique values from the table. Distinct removes all the duplicate records while retrieving from the database.
The syntax for DISTINCT Keyword
SELECT distinct column-name from table-name;
select distinct salary from Emp;

Moreover, & OR operator
AND and OR operators are used with Where clause to make more precise conditions for fetching data from database by combining more than one condition together.
Example
SELECT * from Emp WHERE salary < 10000 AND age > 25
SELECT * from Emp WHERE salary > 10000 OR age > 25

SQL Constraints
SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside the table.

Constraints can be divided into the following two types,
Column level constraints: limits only column data
Table-level constraints: limits whole table data

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

NOT NULL Constraint
NOT NULL constraint restricts a column from having a NULL value. Once NOT NULL constraint is applied to a column, you cannot pass a null value to that column.
Ex. CREATE table Student (s_id int NOT NULL, Name varchar (60), Age int);

UNIQUE Constraint
UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data.
Ex. CREATE table Student (s_id int NOT NULL UNIQUE, Name varchar (60), Age int);

Primary Key Constraint
Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain a null value.
Ex. CREATE table Student (s_id int PRIMARY KEY, Name varchar (60) NOT NULL, Age int);

Foreign Key Constraint
FOREIGN KEY is used to relate two tables. The foreign KEY constraint is also used to restrict actions that would destroy links between tables.
Ex. CREATE table Order_Detail (order_id int PRIMARY KEY,
order_name varchar (60) NOT NULL,
c_id int FOREIGN KEY REFERENCES Customer_Detail(c_id));
On Delete Cascade: This will remove the record from the child table if that value of the foreign key is deleted from the main table.
On Delete Null: This will set all the values in that record of child table as NULL, for which the value of the foreign key is deleted from the main table.

CHECK Constraint

A check constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. It's like condition checking before saving data into a column.
create table Student (s_id int NOT NULL CHECK (s_id > 0),
Name varchar (60) NOT NULL, Age int);

SQL Functions
SQL provides many built-in functions to perform operations on data. These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc. SQL functions are divided into two categories,

Aggregate Functions: -These functions return a single value after calculating from a group of values. Following are some frequently used aggregate functions.
AVG (), COUNT ()

Scalar Functions: -
Scalar functions return a single value from an input value. Following are soe frequently used Scalar Functions.
UCASE ()
UCASE function is used to convert the value of string column to the Uppercase character.

Join in SQL
SQL Join is used to fetch data from two or more tables, which is joined to appear as a single set of data. SQL Join is used for combining column from two or more tables by using values common to both tables. Join Keyword is used in SQL queries for joining two or more tables. Minimum required condition for joining table is (n-1) where n, is a number of tables. A table can also join to itself known as, Self-Join.

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.
Consider the following two tables −
Table 1 – CUSTOMERS Table

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 2 – O'DE'S Table

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables in our SELECT statement as shown below.

```
SQL> SELECT ID, NAME, AGE, AMOUNT
   FROM CUSTOMERS, ORDERS
   WHERE CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

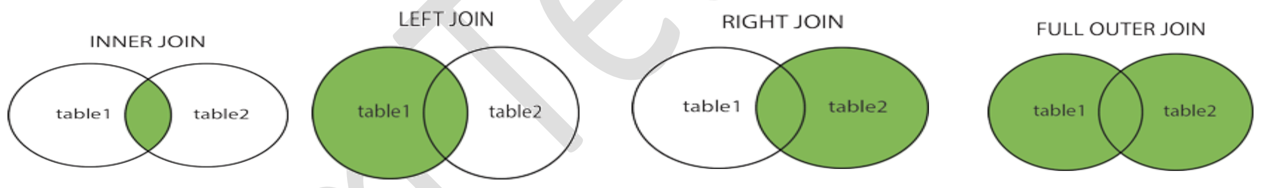
This would produce the following result.

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
| 3 | kaushik  |  23 |   3000 |
| 3 | kaushik  |  23 |   1500 |
| 2 | Khilan   |  25 |   1560 |
| 4 | Chaitali |  25 |   2060 |
+----+----------+-----+--------+
```

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=,!=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal to symbol.

There are several types of joins available in SQL –

- INNER JOIN – returns rows when there is a match in both tables.
- LEFT JOIN – returns all rows from the left table, even if there are no matches in the right table.
- RIGHT JOIN – returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN – returns rows when there is a match in one of the tables.
- SELF JOIN – is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- CARTESIAN JOIN – returns the Cartesian product of the sets of records from the two or more joined tables.



Example: -

Orders

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID

Customers

CustomerID	**CustomerName**	**ContactName**	**Address**	**City**	**PostalCode**	**Country**

Inner Join: -

1. SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

2. SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);

Left Join: -

SELECT Customers.CustomerName, Orders.OrderID
FROM Customers

```
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Right Join: -

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

Full Outer Join: -

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Self-Join: -

```
SELECT A. CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

Query Processing and Optimization

Query Processing

Query Processing is a procedure of transforming a high-level query (such as SQL) into a correct and efficient execution plan expressed in low-level language. A query processing selects a most appropriate plan that is used in responding to a database request. When a database system receives a query for update or retrieval of information, it goes through a series of compilation steps, called execution plan.

Phases are: -

- In the first phase called syntax checking phase, the system parses the query and checks that it follows the syntax rules or not.
- It then matches the objects in the query syntax with the view tables and columns listed in the system table.
- Finally, it performs the appropriate query modification. During this phase, the system validates the user privileges and that the query does not disobey any integrity rules.
- The execution plan is finally executing to generate a response.

So, query processing is a stepwise process.

- The user gives a query request, which may be in QBE or another form. This is first transformed into a standard high-level query language, such as SQL.
- This SQL query is read by syntax analyser so that it can be check for correctness.
- At this step, the syntax analyser uses the grammar of SQL as input and the parser portion of the query processor check the syntax and verify whether the relation and attributes of the requested query are defined in the database.
- At this stage, the SQL query is translated into an algebraic expression using various rules.
- So that the process of transforming a high-level SQL query into a relational algebraic form is called Query Decomposition.
- The relational algebraic expression now passes to the query optimiser. Here optimisation is performed by substituting equivalent expression depends on the factors such that the existence of specific database structures, whether a given file is stored, the presence of different indexes & so on.

- Query optimisation module work in tandem with the join manager module to improve the order in which joins are performed. At this stage, the cost model and several other estimation formulas are used to rewrite the query.
- The modified query is written to utilise system resources to bring the optimal performance.
- The query optimiser then generates an action plan also called an execution plan. These action plans are converted into a query code that are finally executed by a runtime database processor.
- The runtime database processor estimated the cost of each action plan and chose the optimal one for the execution.

Query Analyzer

- The syntax analyser takes the query from the users, parses it into tokens and analyses the tokens and their order to make sure they follow the rules of the language grammar.
- Is an error is found in the query submitted by the user, it is rejected, and an error code together with an explanation of why the query was rejected is a return to the user.

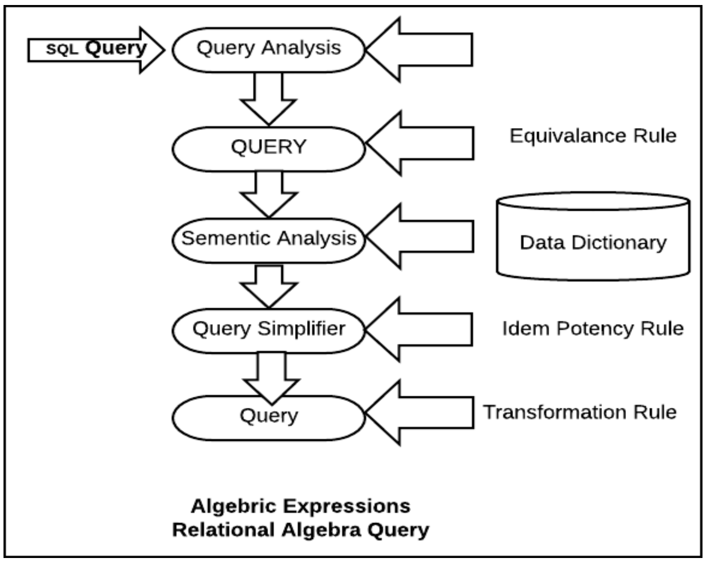
A simple form of the language grammar that could use to implement SQL statement is given bellow:

- QUERY = SELECT + FROM + WHERE
- SELECT = 'SELECT' + <CLOUMN LIST>
- F'OM = 'F'OM' + <TABLE LIST>
- WHE‘E = 'WHE'E' + VALUE1 OP VALUE2
- VALUE1 = VALUE / COLUMN NAME
- VALUE2 = VALUE / COLUMN NAME
- OP = >, <, >=, <=, =, <>

Query Decomposition

The query decomposition is the first phase of the query processing whose aims are to transfer the high-level query into a relational algebra query and to check whether that query is syntactically and semantically correct.

- Thus, the query decomposition starts with a high-level query and transform into a query graph of low-level operations, which satisfy the query.
- The SQL query is decomposed into query blocks (low-level operations), which form the basic unit.
- Hence nested queries within a query are identified as separate query blocks.
- The query decomposer goes through five stages of processing for decomposition into low-level operation and translation into algebraic expressions.



Phases of Query Processing

1) Query Analysis: -

- During the query analysis phase, the query is syntactically analysed using the programming language compiler (parser).
- A syntactically legal query is then validated, using the system catalogue, to ensure that all data objects (relations and attributes) referred to by the query are defined in the database.
- The type specification of the query qualifiers and result is also checked at this stage.
  Let us consider the following query :
  SELECT emp_nm FROM EMPLOYEE WHERE emp_desg>100
  This query will be rejected because the comparison ">100" is incompatible with the data type of emp_desg which is a variable character string.

QUERY TREE NOTATIONS: -

At the end of the query analysis phase, the high-level query (SQL) is transformed into some internal representation that is more suitable for processing. This internal representation is typically a kind of query tree.

A Query Tree is a tree data structure that corresponds expression.

A Query Tree is also called a relational algebra tree.

· The leaf node of the tree, representing the base input relations of the query.
· Internal nodes of the tree representing an intermediate relation, which is the result of applying operation in the algebra.
· The root of the tree representing a result of the query.
· The sequence of operations is directed from ***leaf to root***.

The query tree is executed by executing a

· The resulting relation then replaces the internal.
· The execution is terminated when the root relation for the query.

Example: -

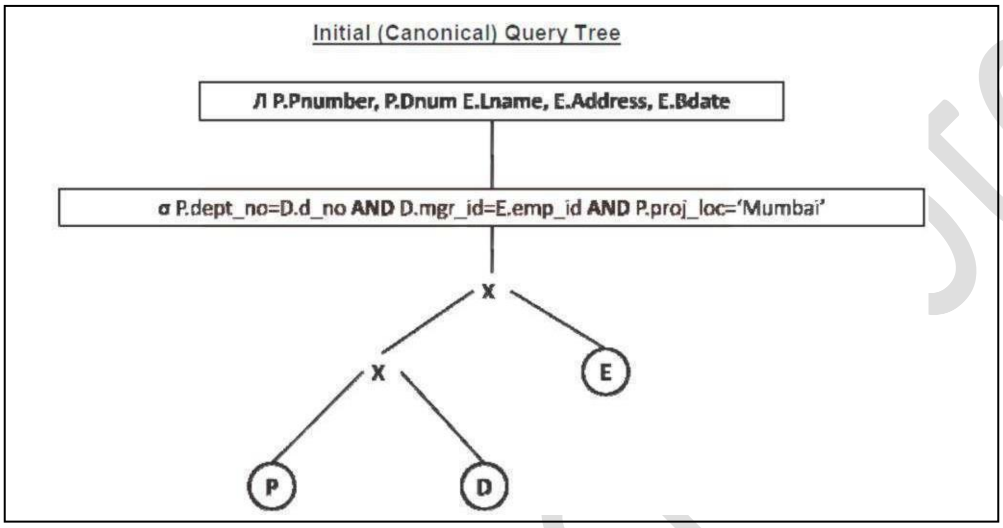
SELECT (P. proj_no, P.dept_no, E.name, E.add, E.dob)
FROM PROJECT P, DEPARTMENT D, EMPLOYEE
WHERE P. dept_no = D.d_no AND D.mgr_id = E.emp_id
Moreover, P. proj_loc = 'Mumbai' ;

Mumbai-PROJ ← σ proj_loc = 'Mumbai' (P'OJECT)
CONT-DEPT ← (Mumbai-PROJ ⋈ dept_no = d_no DEPARTMENT)

PROJ-DEPT-MGR🡐 (CONT-DEPT ⋈ mgr_id=emp_id EMPLOYEE)
RESULT🡐 Π proj_no, dept_no, name, add, dob (P'OJ-DEPT-MGR)

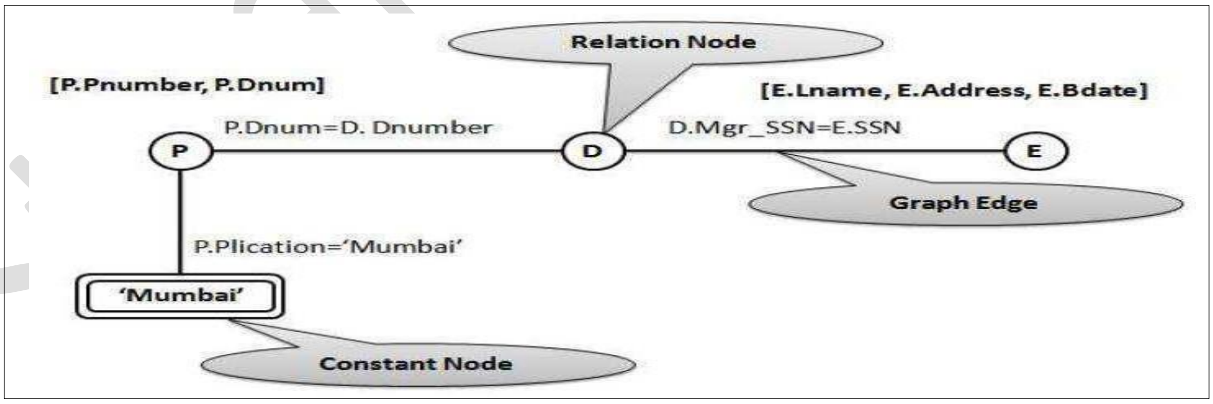
The three relations PROJECT, DEPARTMENT, EMPLOYEE is representing as a leaf nodes P, D and E, while the relational algebra operations of the represented by internal tree nodes.

- The same SQL query can have man different relational algebra expressions and hence many different query trees.
- The query parser typically generates a standard initial (canonical) query tree.


Initial (Canonical) Query Tree
Л P.Pnumber, P.Dnum E.Lname, E.Address, E.Bdate
σ P.dept_no=D.d_no AND D.mgr_id=E.emp_id AND P.proj_loc='Mumbai'
X
X
E
P
D

QUERY GRAPH NOTATIONS

- Query graph is sometimes also used for the representation of a query. In query graph representation, the relations in the query are represented **relation nodes** are displayed as a SINGLE CIRCLE.
- The constant values from the query selection (proj_loc = 'Mumbai') are represented by a **constant node**, displayed as a DOUBLE CIRCLE.
- The **selection and join conditions are represented as a Graph Edge** (e.g. P. dept_no = p.dept_num).
- Finally, the attributes retrieve from the relation are displayed in square brackets (P. proj_num, P.dept_no).



2) Query Normalization: -

- The primary phase of the normalisation is to avoid redundancy. The normalisation phase converts the query into a normalised form that can be more easily manipulated.
- In the normalisation phase, a set of equivalence rules are applied so that the projection and selection operations included on the query are simplified to avoid redundancy.
- The projection operation corresponds to the SELECT clause of the SQL query and the selection operation

correspond to the predicate found in WHERE clause.

- The equivalency transformation rules that are applied to SQL query is shown in the following table, in which UNARYOP means UNARY operation, BINOP means BINARY operation and REL1, REL2, REL3 are the relations.

	Rule Name	Rule Description
1.	Commutative of a UNARY operation	UNARYOP1 UNARYOP2 REL ↔ UNARYOP2 UNARYOP1 REL
2.	Commutative of BINARY operation	REL1 BINOP (REL2 BINOP REL3) ↔ (REL1 BINOP REL2) BINOP REL3
3.	Idempotency of UNARY operations UNARYOP1 UNARYOP2 REL	UNARYOP REL
4.	Distributivity of UNARY operations	UNARYOP1 (REL1 BINOP REL2) ↔ UNARYOP (REL1) BINOP UNARYOP (REL2)
5.	Factorisation ofUNARY operations	UNARIOP (REL1) BINOP UNARYOP (REL2) ↔ UNARYOP (REL1 BINOP REL2)

3) Semantic Analyzer: -

- The objective of this phase of query processing is to reduce the number of predicates.
- The semantic analyser rejects the normalised queries that are incorrectly formulated.
- A query is incorrectly formulated if components do not contribute to the generation of the result. This happens in the case of missing join specification.
- A query is contradictory if its predicate cannot satisfy by any tuple in the relation. The semantic analyser examines the relational calculus query (SQL) to make sure it contains only data objects that are a table, columns, views, indexes that are defined in the database catalogue.
- It makes sure that each object in the query is referenced correctly according to its data type.
- In the case of missing join specifications, the components do not contribute to the generation of the results, and thus, a query may be incorrectly formulated.
- A query is contradictory if its predicates cannot be satisfied by any of the tuples.
  For example: -
  (emp_des = 'Programmer' A emp_des = 'Analyst')
  As an employee cannot be both 'Programmer' and 'Analyst' simultaneously, the above predicate on the EMPLOYEE relation is contradictory.

Example of Correctness and Contradictory: -

Incorrect Formulated: - (Missing Join Specification)
SELECT p. projno, p.proj_location
FROM project p, viewing v, department d
WHERE d. dept_id = v.dept_id AND d.max_budj >= 8000
AND d.mgr = 'Methew'
Contradictory: - **(Predicate cannot satisfy)**
SELECT p. proj_no, p.proj_location
FROM project p, cost_proj c, department d
WHERE d.max_budj >80000 AND d.max_budj < 50000 AND
d.dept_id = v. dept_id AND v.proj_no = p.proj_no

4.Query Simplifier: -

The objectives of this phase are to detect redundant qualification, eliminate common sub-expressions and transform sub-graph to semantically equivalent but easier and efficiently computed form.

Why simplify? :-
Commonly integrity constraints view definitions, and access restrictions are introduced into the graph at this stage of analysis so that the query must be simplified as much as possible.
Integrity constraints define constants which must hold for all state of the database, so any query that contradicts integrity constraints must be avoided and can be rejected without accessing the database.

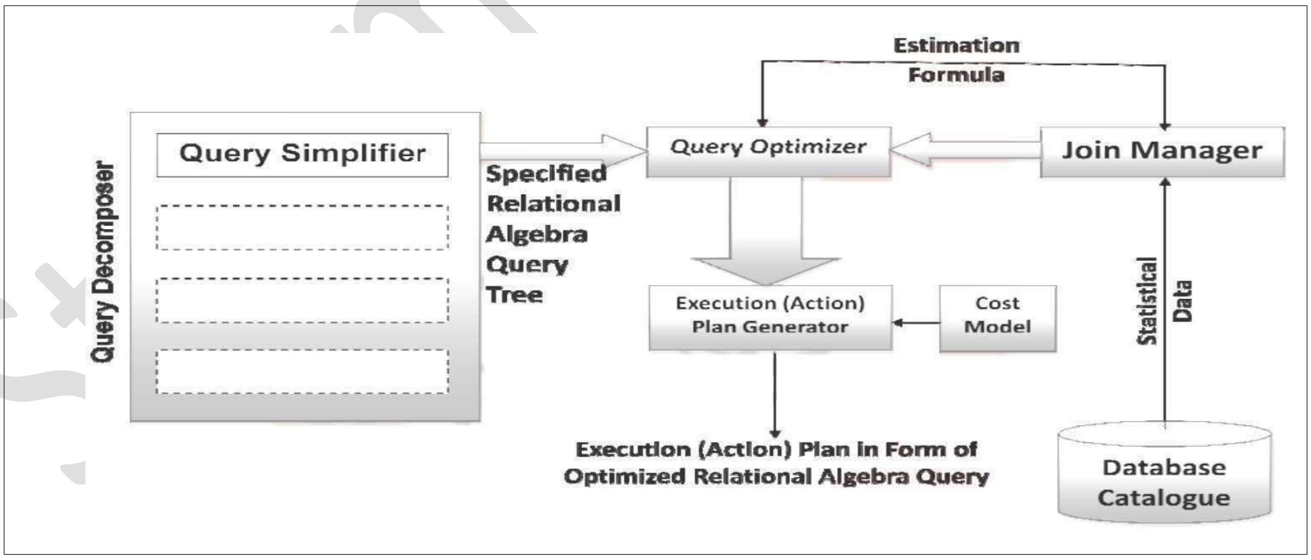
The final form of simplification is obtaining by applying **idempotency rules** of Boolean algebra.

	Description	Rule Format
1.	**PRED** AND **PRED** = **PRED**	P A (P) = P
2.	**PRED** AND TRUE = **PRED**	P v TRUE = P
3.	**PRED** AND FALSE = FALSE	P A FALSE = FALSE
4.	**PRED** AND NOT(**PRED**) = FALSE	P A (~P) = FALSE
5.	**PRED1** AND **(PRED1** OR **PRED2)** = **PRED1**	P1 A (P1 v P2) = P1
6.	**PRED** OR **PRED** = **PRED**	P v (P) = P
7.	**PRED** OR TRUE = TRUE	P v TRUE = TRUE
8.	**PRED** OR FALSE = **PRED**	P v FALSE = P
9.	**PRED** OR NOT(**PRED**) = TRUE	P v (~P) = TRUE
10.	**PRED1** OR (**PRED1** AND **PRED2**) = **PRED1**	P1 v (P1 A P2) = P1

5) Query Restructuring: -

- In the final stage of the query decomposition, the query can be restructured to give a more efficient implementation.
- Transformation rules are used to convert one relational algebra expression into an equivalent form that is more efficient.
- The query can now be regarded as a relational algebra program, consisting of a series of operations on the relation.

Query Optimization


Estimation Formula
Query Simplifier
Query Decomposer
Specified Relational Algebra Query Tree
Query Optimizer
Join Manager
Execution (Action) Plan Generator
Cost Model
Statistical Data
Execution (Action) Plan in Form of Optimized Relational Algebra Query
Database Catalogue

Query Optimization Phases

The primary goal of query optimisation is of choosing an efficient execution strategy for processing a query.

- The query optimiser attempts to minimise the use of specific resources (mainly the number of I/O and CPU time) by selecting the best execution plan (access plan).
- A query optimisation starts during the validation phase by the system to validate the user has

appropriate privileges.
- Now an action plan is generating to perform the query.

The Relational algebra query tree is generated by the query simplifier module of query decomposer.
- Estimation formulas used to determine the cardinality of the intermediate result table.
- A cost Model
- Statistical data from the database catalogue.

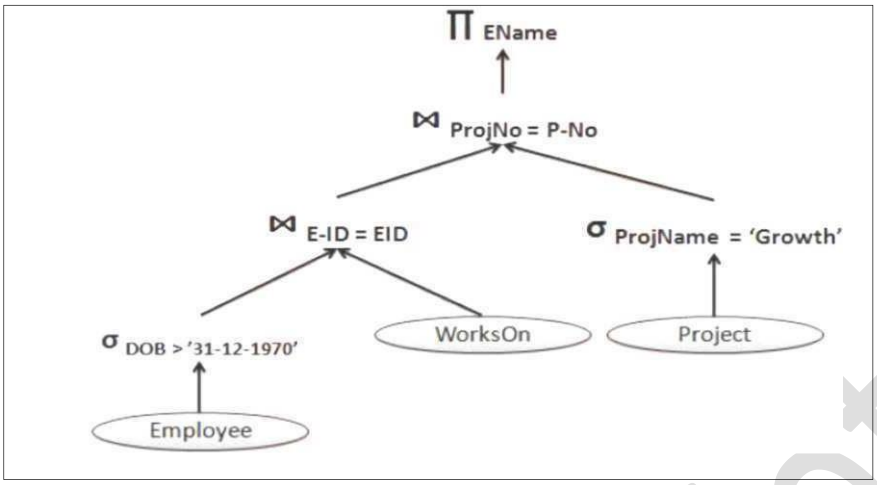
The output of the query optimiser is the execution plan in the form of optimised relational algebra query.
- A query typically has many possible execution strategies, and the process of choosing a suitable one for processing a query is known as Query Optimization.
- The fundamental issues in Query Optimization are:
  - How to use available indexes.
  - How to use memory to accumulate information and perform immediate steps such as sorting.
  - How to determine the order in which joins should be performed.
- The term query optimisation does not mean always giving an optimal (best) strategy as the execution plan.
- It is just a responsibly efficient strategy for execution of the query.
- The decomposed query block of SQL is translating into an equivalent extended relational algebra expression and then optimised.

There are two main techniques for implementing Query Optimization:
The **first** technique is based on ***Heuristic Rules*** for ordering the operations in a query execution strategy. The **second** technique involves the ***systematic estimation*** of the cost of the different execution strategies and choosing the execution plan with the *lowest cost*.
- Semantic query optimisation is used with the combination with the heuristic query transformation rules.
- It uses constraints specified on the database schema such as unique attributes and other more complex constraints, to modify one query into another query that is more efficient to execute.

1. Heuristic Rules: -
- The heuristic rules are used as an optimisation technique to modify the internal representation of the query. Usually, heuristic rules are used in the form of query tree of query graph data structure, to improve its performance.
- One of the main heuristic rules is to apply SELECT operation before applying the JOIN or other BINARY operations.
- This is because the size of the file resulting from a binary operation such as JOIN is usually a multi-value function of the sizes of the input files.
- The SELECT and PROJECT reduced the size of the file and hence, should be applied before the JOIN or other binary operation.
- Heuristic query optimiser transforms the initial (canonical) query tree into final query tree using equivalence transformation rules. This final query tree is efficient to execute.

For example, consider the following relations:
Employee (EName, EID, DOB, EAdd, Sex, ESalary, EDeptNo)
Department (DeptNo, DeptName, DeptMgrID, Mgr_S_date)
DeptLoc (DeptNo, Dept_Loc)
Project (ProjName, ProjNo, ProjLoc, ProjDeptNo)
WorksOn (E-ID, P-No, Hours)
Dependent (E-ID, DependName, Sex, DDOB, Relation)
Now let us consider the query in the above database to find the name of employees born after 1970 who work on a project named 'Growth'.
SELECT EName

FROM Employee, WorksOn, Project
WHE˝E ProjName = 'Growth' AND ProjNo = P-No
AND EID = E-ID AND DOB > '31-12-1970';



General Transformation Rules: -
Transformation rules are used by the query optimiser to transform one relational algebra expression into an equivalent expression that is more efficient to execute.

- A relation is considering as equivalent of another relation if two relations have the same set of attributes in a different order but representing the same information.
- These transformation rules are used to restructure the initial (canonical) relational algebra query tree attributes during query decomposition.

1. Cascade of σ :-
σ c1 AND c2 AND …AND cn (') = σ c1 (σ c2 (… (σ cn ('))…))
2. Commutativity of σ :-
σ C1 (σ C2 (')) = σ C2 (σ C1 ('))
3. Cascade of Л :-
Л List1 (Л List2 (… (Л List n ('))…)) = Л List1 (')
4. Commuting σ with Л :-
Л A1, A2,A3…An (σ C (')) = σ C (Л A1,A2,A3…An ('))
5. Commutativity of ⋈ AND x :-
R ⋈ c S = S ⋈ c R
R x S = S x R
6. Commuting σ with ⋈ or x :-
If all attributes in selection condition c involved only attributes of one of the relation schemas (R).
σ c (R ⋈ S) = (σ c (')) ⋈ S
Alternatively, selection condition c can be written as (c1 AND c2) where condition c1 involves only attributes of R and condition c2 involves only attributes of S then:
σ c (R ⋈ S) = (σ c1 (')) ⋈ (σ c2 (S))

7. Commuting Л with ⋈ or x:-

- The projection list L = {A1, A2,..An,B1,B2,…Bm}.
- A1…An attribute of ' and B1…Bm attributes of S.
- Join condition C involves only attributes in L then:
- ЛL (R ⋈ c S) = (ЛA1,…An (')) ⋈c (ЛB1,…Bm(S))

8. Commutativity of SET Operation: -
- R ∪ S = S ∪ R
- R ß S = S ß R
Minus (R-S) is not commutative.

9. Associatively of ⋈, x, ß, and ∪:-
- If ∅ stands for any one of this operation throughout the expression then:
(R ∅ S) ∅ T = R ∅ (S ∅ T)
10. Commutativity of σ with SET Operation: -
- If ∅ stands for any one of three operations (∪, ß,and-) then :
σ c (R ∅ S) = (σ c (‘)) ∪ (σ c (S))
Л c (R ∅ S) = (Л c (‘)) ∪ (Лc (S))
11. The Л operation commute with ∪:-
Л L (R ∪ S) = (Л L(‘)) ∪ (Л L(S))
12. Converting a (σ, x) sequence with ∪
(σ c (‘ x S)) = (‘ ⋈ c S)

Heuristic Optimization Algorithm: -
The Database Management System use Heuristic Optimization Algorithm that utilises some of the transformation rules to transform an initial query tree into an optimised and efficient executable query tree.

- The steps of the heuristic optimisation algorithm that could be applied during query processing and optimization are:
  Step-1: -
- Perform SELECT operation to reduce the subsequent processing of the relation:
- Use the transformation rule 1 to break up any SELECT operation with conjunctive condition into a cascade of SELECT operation.

Step-2: -

- Perform commutativity of SELECT operation with other operation at the earliest to move each SELECT operation down to query tree.
- Use transformation rules 2, 4, 6 and 10 concerning the commutativity of SELECT with other operation such as unary and binary operations and move each select operation as far down the tree as is permitted by the attributes involved in the SELECT condition. Keep selection predicates on the same relation together.

Step-3: -

- Combine the Cartesian product with a subsequent SELECT operation whose predicates represents the join condition into a JOIN operation.
- Use the transformation rule 12 to combine the Cartesian product operation with the subsequent SELECT operation.

Step-4: -

- Use the commutativity and associativity of the binary operations.
- Use transformation rules 5 and 9 concerning commutativity and associativity to rearrange the leaf nodes of the tree so that the leaf node with the most restrictive selection operation is executed first in the query tree representation. The most restrictive SELECT operation means:
- Either the one that produces a relation with the fewest tuples or with the smallest size.
- The one with the smallest selectivity.

Step-5: -

- Perform the projection operation as early as possible to reduce the cardinality of the relationship and the subsequent processing of the relation and move the Projection operations as far down the query tree as possible.
- Use transformation rules 3, 4, 7 and 10 concerning the cascading and commuting of projection operations with other binary operation. Break down and move the projection attributes down the tree as far as needed. Keep the projection attributes in the same relation together.

Step-6: -

- Compute common expression once.

- Identify sub-tree that represent a group of operations that can be executed by a single algorithm.

2. Cost Estimation in Query Optimization: -

- The main aim of query optimisation is to choose the most efficient way of implementing the relational algebra operations at the lowest possible cost.
- Therefore, the query optimiser should not depend solely on heuristic rules, but it should also estimate the cost of executing the different strategies and find out the strategy with the minimum cost estimate.
- The method of optimising the query by choosing a strategy those result in minimum cost is called cost-based query optimisation.
- The cost-based query optimisation uses the formula that estimates the cost for some options and selects the one with the lowest cost and the most efficient to execute.
- The cost functions used in query optimisation are estimates and not exact cost functions.
- The cost of an operation is heavily dependent on its selectivity, that is, the proportion of select operation(s) that forms the output.
- In general, the different algorithms are suitable for low or high selectivity queries. For the query optimiser to choose the suitable algorithm for an operation an estimate of the cost of executing that algorithm must be provided.
- The cost of an algorithm depends on cardinality of its input.
- To estimate the cost of different query execution strategies, the query tree is viewed as containing a series of basic operations which are linked to perform the query.
- It is also essential to know the expected cardinality of an operation’s output because of this form the input to the next operation.

Cost Components of Query Execution: -

- The success of estimating the size and cost of standard relational algebra operations depends on the amount the accuracy of statistical data information stored with DBMS.

The cost of executing the query includes the following components: -

- Access cost to secondary storage.
- Storage cost.
- Computation cost.
- Memory uses cost.
- Communication cost.

1. Access cost to secondary storage: -

Access cost is the cost of searching for reading and writing data blocks (containing the number of tuples) that reside on secondary storage, mainly on disk of the DBMS.

The cost of searching for tuples in the database relations depends on the type of access structures on that relation, such as ordering, hashing and primary or secondary indexes.

2. Storage cost: -

The storage cost is of storing any intermediate relations that are generated by the executing strategy for the query.

3. Computation cost: -

_ Computation cost is the cost of performing in-memory operations on the data buffers during query execution.

_ Such operations contain searching for and sorting records, merging records for a join and performing computation on a field value.

4. Memory uses cost: -

_ Memory uses cost a cost about the number of memory buffers needed during query execution.

5. Communication cost: -

_ It is the cost of transferring query and its results from the database site to the site of the terminal of query organisation.

_ Out of the above five cost components; the most important is the secondary storage access cost.

_ The emphasis of the cost minimisation depends on the **size** and **type** of database applications.
_ For example, in the smaller database the emphasis is on the minimising computing cost as because most of the data in the files involved in the query can be completely stored in the main memory. For a large databaprimarythe main emphasis is on minimising the access cost to the secondary device.

- For the distributed database, the communication cost is minimised as because many sites are involved in the data transfer.
- To estimate the cost of various execution strategies, we must keep track of any information that is needed for the cost function. This information may be stored in the database catalogue, where the query optimizer accesses it.
- Typically, the DBMS is expected to hold the following types of information in its system catalogue.

· The number of tuples in relation to R [nTuples(R)].
· The average record size in relation R.
· The number of blocks required to store relation R as [nBlocks(R)].
· The blocking factors in relation R (that is the number of tuples of R that fit into one block) as [bFactor(R)].
· Primary access method for each file.
· Primary access attributes for each file.
· The number of levels of each multilevel index I (primary, secondary or clustering) as [nLevelsA(I)].
· The number of first level index blocks as [nBlocksA (I)].
· The number of distinct values that appear for attribute A in relation R as [nDistinctA(R)].
· The minimum and maximum possible values for attribute A in relation R as [minA(R), maxA(R)].
· The selectivity of an attribute, which is the fraction of records satisfying an equality condition on the attribute.
· The selection cardinality of given attribute A in relation R as [SCA(R)]. The selection cardinality is the average number of tuples that satisfied an equality condition on attribute A.

- **Cost functions for SELECT Operation: -**

➢ Linear Search: -
  · [nBlocks(R)/2], if the record is found.
  · [nBlocks(R)], if no record satisfied the condition.
➢ Binary Search: -
  · [log2(nBlocks(R))], if equality condition is on key attribute, because SCA(R) = 1 this case.
  · [log2(nBlocks(R))] + [SCA(R)/bFactor(R)] – 1, otherwise.
➢ Using primary index or hash key to retrieve a single record: -
  · 1, assuming no overflow
➢ Equity condition on Primary key: -
  · [nLevelA(I) + 1]
➢ Equity condition on Non-Primary key: -
  · [nLevelA(I) + 1] + [nBlocks(R)/2]
➢ Using inequality condition on a secondary index (B+ Tree): -
  · [nLevelA(I) + 1] + [nLfBlocksA(I)/2] + [nTuples(R)/2]
➢ Equity condition on clustering index: -
  · [nLevelA(I) + 1] + [SCA(R)/bFactor(R)]
➢ Equity condition on non-clustering index: -
  · [nLevelA(I) + 1] + [SCA(R)]

Example of Cost Estimation for SELECT Operation: -
· Let us consider the relation EMPLOYEE having following attributes: -
EMPLOYEE (EMP-ID, DEPT-ID, POSITION, SALARY)
· Let us consider the following assumptions: -
o There is a hash index with no overflow on primary key attribute EMP-ID.
o There is a clustering index on foreign key attribute DEPT-ID.
o There is a B+-Tree index on the SALARY attribute.
· Let us also assume that the EMPLOYEE relation has the following statistics in the system catalog:

nTuples(EMPLOYEE) = 6

nTuples(EMPLOYEE)	=	6,000
bFactor(EMPLOYEE)	=	60
nBlocks(EMPLOYEE)	= =	nTuples(EMPLOYEE) / bFactor(EMPLOYEE) 6,000 / 60 = 100
nDistinctDEPT-ID(EMPLOYEE)	=	1,000
SCDEPT-ID(EMPLOYEE)	= =	nTuples(EMPLOYEE) / nDistinctDEPT-ID(EMPLOYEE) 6,000 / 1,000 = 6
nDistinctPOSITION(EMPLOYEE)	=	20
SC POSITION(EMPLOYEE)	= =	nTuples(EMPLOYEE) / nDistinctPOSITION(EMPLOYEE) 6,000 / 20 = 300
nDistinctSALARY(EMPLOYEE)	=	1,000
SCSALARY(EMPLOYEE)	= =	nTuples(EMPLOYEE) / nDistinctSALARY(EMPLOYEE) 6,000 / 1,000 = 6
minSALARY(EMPLOYEE)	=	20,000
maxSALARY(EMPLOYEE)	=	80,000
nLevelsDEPT-ID(I)	=	2
nLevelsSALARY(I)	=	2
nLfBlocksSALARY(I)	=	50

· Selection 1: - б EMP-ID = ‘106519’ (EMPLOYEE)
· Selection 2: - б POSITION = ‘MANAGE‟ (EMPLOYEE)
· Selection 3: - б DEPT-ID = ‘SAP-04’ (EMPLOYEE)
· Selection 4: - б SALA‘Y = 30,000 (EMPLOYEE)
· Selection 5: - б POSITION = ‘MANAGE‟ Λ DEPT-ID = ‘SPA-04’ (EMPLOYEE)
Now we will choose the query execution strategies by comparing the cost as follows:

Selection -1	The selection operation contains an equality condition on the primary key EMP-ID of the relation EMPLOYEE. Therefore, as the attribute EMP-ID is hashed we can use the strategy 3 to estimate the cost as one block. The estimated cardinality of the result relation is **SC EMP-ID (EMPLOYEE) = 1**.
Selection -2	The attribute in the predicate is the non-key, non-indexed attribute. Therefore, we can improve on the linear search method, giving an estimated cost of 100 blocks. The estimated cardinality of the result relation is **SC POSITION (EMPLOYEE) = 300**.
Selection -3	The attribute in the predicate is a foreign key with a clustering index. Therefore, we can use strategy 7 to estimate the cost as **(2 + (6/30)) = 3 blocks**. The estimated cardinality of result relation is **SC DEPT-ID (EMPLOYEE) = 6**.
Selection -4	The predicate here involves a range search on the SALARY attribute, which has the B^+-Tree index. Therefore we can use the strategy 6 to estimate the cost as (2 + (50/2) + (6,000/2)) = 3027 blocks. Thus, the linear search strategy is used in this case, the estimated cardinality of the result relation is SC SALARY(EMPLOYEE) = [6000*(8000-2000*2)/ (8000-2000)] = 4000.

Selection -5	While we are retrieving, each tuple using the clustering index, we can check whether they satisfied the first condition (POSITION = 'MANAGE''). We know that estimated cardinality of the second condition **SC DEPT-ID (EMPLOYEE) = 6**. Let us assume that this intermediate condition is S. then the number of distinct values of POSITION in S can be estimated as **[(6 + 20)/2] = 9**. Let us apply now the second condition using the clustering index on DEPT-ID, which has an estimated cost of 3 blocks. Thus, the estimated cardinality of the result relation will be SC POSITION (S) = 6/9 = 1, which would be correct if there is one manager for each branch.

Join operation is the most time-consuming operation to process. An estimate for the size (number of tuples) of the file that results after the JOIN operation is required to develop reasonably accurate cost functions for JOIN operations.

The JOIN operations define the relation containing tuples that satisfy a specific predicate F from the Cartesian product of two relations R and S.

Following table shows the different strategies for JOIN operations.

Strategies	**Cost Estimation**
Block nested-loop JOIN	**a) nBlocks(R) + (nBlocks(R) * nBlocks(S)) If the buffer has only one block. b) nBlocks(R) + [nBlocks(S) * (nBlocks(R)/(nBuffer-2))] If (nBuffer-2) blocks is there for R c) nBlocks(R) + nBlocks(S) If all blocks of R can be read into the database buffer**
Indexed nested-loop JOIN	**a) nBlocks(R) + nTuples(R) * (nLevelA(I) + 1) If join attribute A in S is a primary key b) nBlocks(R) + nTuples(R) * (nLevelA(I) + [SCA(R) / bFactor(R)]) If clustering index I is on attribute A.**
Sort-merge JOIN	**a) nBlocks(R) * [log2nBlocks(R)] + nBlocks(S) * [log2nBlocks(R)] For Sort b) nBlocks(R) + nBlocks(S) For Merge**
Hash JOIN	**a) 3 (nBlocks(R) + nBlocks(S)) If Hash index is in memory b) 2 (nBlocks(R) + nBlocks(S)) * [log (nBlocks(S)) - 1] + nBlocks(R) + nBlocks(S)**
Sort-merge JOIN	**a) nBlocks(R) * [log2nBlocks(R)] + nBlocks(S) * [log2nBlocks(R)] For Sort b) nBlocks(R) + nBlocks(S) For Merge**
Hash JOIN	**a) 3 (nBlocks(R) + nBlocks(S)) If Hash index is in memory b) 2 (nBlocks(R) + nBlocks(S)) * [log (nBlocks(S)) - 1] + nBlocks(R) + nBlocks(S)**

Unit IV

Normalisation of Database

Database Normalizations is a technique of organising the data in the database. Normalisation is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables.

Normalisation is used for mainly two purposes,

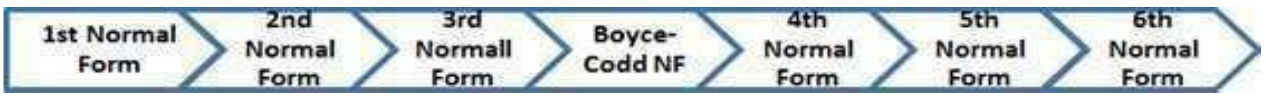
Eliminating redundant(useless) data.
perusalEnsuring data dependencies make sense, i.e. data is logically stored.

Problem Without Normalization

Without Normalization, it becomes difficult to handle and update the database, without facing data loss. Insertion, Updating and Deletion Anomalies are very frequent if Database is not Normalized. To understand these anomalies let us take an example of **Student** table.

S_id	S_Name	S_Address	Subject_opted
401	Adam	Noida	Bio
402	Alex	Panipat	Maths
403	Stuart	Jammu	Maths
404	Adam	Noida	Physics

- **Updating Anomaly:** To update the address of a student who occurs twice or more than twice in a table, we will have to update the **Address** column in all the rows, else data will become inconsistent.
- **Insertion Anomaly:** Suppose for a new admission, we have a Student id(S_id), name and address of a student but if the student has not opted for any subjects yet then we must insert **NULL** there, leading to Insertion Anomaly.
- **Deletion Anomaly:** If (S_id) 401 has only one subject and temporarily he drops it, when we delete that row, entire student record will be deleted along with it.
  Theory of Data Normalization in SQL is still being developed further. For example, there are discussions even on 6th Normal Form. However, in most practical applications normalisation achieves its best in 3rd Normal Form. The evolution of Normalization theories is illustrated below-



Functional Dependencies

A functional dependency is a relationship between two attributes. Typically, between the PK and other non-key attributes within the table. For any relation R, attribute Y is functionally dependent on attribute X (usually the PK), if for every valid instance of X, that value of X uniquely determines the value of Y.

X ———-> Y

The left-hand side of the FD is called the determinant, and the right-hand side is the dependent.

Examples:

SIN ———-> Name, Address, Birthdate

SIN determines names and address and birthdays. Given SIN, we can determine any of the other attributes within the table.

Sin, Course ———> Date-Completed

Sin and Course determine date completed. This must also work for a composite PK.

ISBN ———-> Title

ISBN determines the title.

Various Types of Functional Dependencies are –

- Single Valued Functional Dependency
- Fully Functional Dependency
- Partial Functional Dependency
- Transitive Functional Dependency
- Trivial Functional Dependency
- Non-Trivial Functional Dependency
  - Complete Non-Trivial Functional Dependency
  - Semi Non-Trivial Functional Dependency

Single Valued Functional Dependency –

The database is a collection of related information in which one information depends on another information. The information is either single-valued or multi-valued. For example, the name of the person or his / her date of birth is single-valued facts. However, the qualification of a person is a multivalued fact.

A simple example of single value functional dependency is when A is the primary key of an entity (e.g. SID), and B is some single-valued attribute of the entity (e.g. Sname). Then, A → B must always hold.

CID	SID	Sname
C1	S1	A
C1	S2	A
C2	S1	A
C3	S1	A

SID → Sname

S1	A
S1	A
S1	A

Sname → SID X

A	S1
A	S2

For every SID, there should be a unique name (X → Y)

Definition: Let R be the relational schema and X, Y be the set of attributes over R. t1, t2 be any tuples of R. X → Y exists in relation R only if t1.X = t2.X then t1.Y = t2.Y

If the condition fails – then the dependency is not there.

Fully Functional Dependency

In a relation R, an attribute Q is said to be fully functional dependent on attribute P, if it is functionally dependent on P and not functionally dependent on any proper subset of P. The dependency P → Q is left reduced, there is no extraneous attributes in the left-hand side of the dependency.

If AD → C, is a fully functional dependency, then we cannot remove A or D., I.e. C is fully functionally dependent on AD. If we can remove A or D, then it is not fully functional dependency.
Another Example, Consider the following Company Relational Schema,
EMPLOYEE

ENAME	SSN(P.K)	BDATE	ADDRESS	NUMBER

DEPARTMENT

DNAME	DNUMBER (P.K)	DMGRSSN (F.K)

DEPT_LOCATIONS

DNUMBER (P.k)	DLOCATION (P.K)

PROJECT

PNAME	PNUMBER	LOCATION	DNUM

WORKS_ON

SSN (P.K)	PNUMBER (P.K)	HOURS

{SSN, PNUMBER} → HOURS is a full FD since neither SSN → HOURS
nor NUMBER → HOURS hold

{SSN, PNUMBER} → ENAME is not a full FD (it is called a partial dependency) since SSN → ENAME also holds.
Partial Functional Dependency –

A Functional Dependency in which one or more non-key attributes are functionally depending on the part of the primary key is called partial functional dependency. or
where the determinant consists of critical attributes, but not the entire primary key, and the determined consist of non-key attributes.
For example, consider a Relation R (A, B, C, D, E) having
FD: AB → CDE where PK is AB.

Then, {A → C; A → D; A → E; B → C; B → D; B → E} all are Partial Dependencies.

Transitive Dependency –

Given a relation R (A, B, C) then dependency like A–>B, B–>C is a transitive dependency, since A–>C is implied.

In the above Figure
SSN --> DMGRSSN is a transitive FD
{since SSN --> DNUMBER and DNUMBER --> DMGRSSN hold}

SSN --> NAME is non-transitive FD since there is no set of attributes X
where SSN --> X and X --> ENAME.
Trivial Functional Dependency –
Some functional dependencies are said to be trivial because they are satisfied by all relations. Functional dependency of form A–>B is trivial if B subset= A. or

A trivial Functional Dependency is the one where RHS is a subset of LHS.
Example, A-->A is satisfied by all relations involving attribute A.
SSN-->SSN
PNUMBER-->PNUMBER
SSN PNUMBER -->PNUMBER
SSN PNUMBER --> SSN PNUMBER

Non-Trivial Functional Dependency –

Non-Trivial Functional Dependency can be categorised into –

- Complete Non-Trivial Functional Dependency
- Semi Non-Trivial Functional Dependency

Complete Non-Trivial Functional Dependency –
A Functional Dependency is entirely non-trivial if none of the RHS attributes is part of the LHS attributes.

Example, SSN --> Ename,
PNUMBER --> PNAME
PNUMBER--> BDATE X
Semi Non-Trivial Functional Dependencies – A Functional Dependency is semi non-trivial if at least one of the RHS attributes are not part of the LHS attributes.

{TRIVIAL + NONTRIVIAL}
Question 1:

A	B	C
1	1	1
1	2	1
2	1	2
2	2	3

Identify Non-Trivial Functional Dependency?
Solution:

S.NO	Dependencies	Non-Trivial FD?
1	A→B	×
2	A→C	×
3	A→BC	×
4	B→A	×
5	B→C	×
6	B→AC	×
7	C→A	√
8	C→B	×
9	C→AB	×
10	AB→C	√
11	BC→A	√
12	AC→B	×

A→B is not a non-trivial FD because, for 2, it has two outputs. i.e 2→2 and 2→3.
for AB→C, 11→1, 12→1, 21→2, 22→3, so Non-trivial.

Question 2: R (A B C D) AB {Candidate Key} A→C B→D. Where is the redundancy existing?
Solution: (A C) and (B D) is suffering from redundancy.
Question 3: Consider a relation with schema R (A, B, C, D) and FDs {AB -> C, C -> D, D -> A}. a. What are some of the nontrivial FDs that can be inferred from the given FDs?

Some examples:

C -> ACD
D -> AD
AB -> ABCD
AC -> ACD
BC -> ABCD
BD -> ABCD
CD -> ACD
ABC -> ABCD
ABD -> ABCD
BCD -> ABCD

Inference Rules for Functional Dependencies –
Armstrong's Inference Rules –
Let A, B and C and D be arbitrary subsets of the set of attributes of the giver relation R, and let AB be the union of A and B. Then, ⇒→
Primary
Reflexivity:
If B is subset of A, then A → B

Augmentation:
If A → B, then AC → BC

Transitivity:
If A → B and B → C, then A → C.
Projectivity or Decomposition Rule:
If A → BC, Then A → B and A → C
Proof:
Step 1: A → BC (GIVEN)
Step 2: BC → B (Using Rule 1, since B ⊆ BC)
Step 3: A → B (Using Rule 3, on step 1 and step 2)

Secondary Rule

Union or Additive Rule:
If A→B, and A→C Then A→BC.
Proof:
Step 1: A → B (GIVEN)
Step 2: A → C (given)
Step 3: A → AB (using Rule 2 on step 1, since AA=A)
Step 4: AB → BC (using rule 2 on step 2)
Step 5: A → BC (using rule 3 on step 3 and step 4)

Pseudo Transitive Rule:
If A → B, DB → C, then DA → C
Proof:
Step 1: A → B (Given)
Step 2: DB → C (Given)
Step 3: DA → DB (Rule 2 on step 1)
Step 4: DA → C (Rule 3 on step 3 and step 2)'

These are not commutative as well as associative.
i.e. if X → Y then
Y → X x (not possible)

Composition Rule:
If A → B, and C → D, then AC → BD.

Self Determination Rule:
A → A is a self-determination rule.
Let S be the set of functional dependencies that are specified on relation schema R. Numerous other dependencies can be inferred or deduced from the functional dependencies in S.

Example:

Let S = {A → B, B → C}

A multivalued dependency occurs when the presence of one or more rows in a table implies the presence of one or more other rows in that same table. Put another way, two attributes (or columns) in a table are independent of one another, but both depend on a third attribute. A multivalued dependency prevents the normalization standard Fourth Normal Form (4NF).

Functional dependency vs. Multivalued dependency
To understand this, let's revisit what a functional dependency is.

Remember that if an attribute X uniquely determines an attribute Y, then Y is functionally dependent on X. This is written as X -> Y. For example, in the Students table below, the Student_Name determines the Major:

Students

Student_Name	Major
Ravi	Art History
Beth	Chemistry

This functional dependency can be written: Student_Name -> Major. Each Student_Name determines exactly one Major, and no more.

Now, perhaps we also want to track the sports these students take. We might think the easiest way to do this is just to add another column, Sport:

Students

Student_Name		Major Sport
Ravi	Art History	Soccer
Ravi	Art History	Volleyball
Ravi	Art History	Tennis
Beth	Chemistry	Tennis
Beth	Chemistry	Soccer

The problem here is that both Ravi and Beth play multiple sports. We need to add a new row for every additional sport.

This table has introduced a multivalued dependency because the major and the sport are independent of one another, but both depend on the student.

Note that this is a very simple example and easily identifiable — but this could become a problem in an extensive, complex database.

A multivalued dependency is written X ->-> Y. In this case:

Student_Name ->-> Major
Student_Name ->-> Sport

This is read as "Student_Name multidetermined Major" and "Student Name multidetermined Sport."

A multivalued dependency always requires at least three attributes because it consists of at least two attributes that are dependent on a third.

Multivalued dependency and normalization
A table with a multivalued dependency violates the normalization standard of Fourth Normal Form (4NK) because it creates unnecessary redundancies and can contribute to inconsistent data. To bring this up to 4NF, we can break this into two tables.

The table below now has a functional dependency of Student_Name -> Major, and no multi dependencies:

Students & Majors

Student_Name	Major
Ravi	Art History
Ravi	Art History
Ravi	Art History
Beth	Chemistry
Beth	Chemistry

While this table also has a single functional dependency of Student_Name -> Sport:

Students & Sports

Student_Name	Sport
Ravi	Soccer
Ravi	Volleyball
Ravi	Tennis
Beth	Tennis
Beth	Soccer

Normalisation is often addressed by simplifying complex tables so that they contain information related to a single idea or theme, rather than trying to make a single table contain too much disparate information.

Numerical on Functional Dependency: -
1. Let R= (A, B, C, D, E, F) be a relation scheme with the following dependencies: C->F, E->A, EC->D, A->B. Which of the following is a key for R?
(a) CD (b) EC (c) AE (d) AC

Ans: option (b)
Explanation:
Find the closure set of all the options given. If any closure covers all the attributes of the relation R then that is the key.

2. Consider a relation scheme R = (A, B, C, D, E, H) on which the following functional dependencies hold: {A–>B, BC–>D, E–>C, D–>A}. What are the candidate keys of R?
(a) AE, BE
(b) AE, BE, DE
(c) AEH, BEH, BCH
(d) AEH, BEH, DEH

Ans: option (d)
Explanation:
As explained in question 1, if any closure includes all attributes of a table then it becomes the candidate key.
Closure of AEH = AEHB {A->B}
= AEHBC {E->C}
= AEHBCD {BC->D}
GATE-2005(IT)

5. In a schema with attributes A, B, C, D and E, following set of functional dependencies are given:
A->B
A->C
CD->E
B->D
E->A
Which of the following functional dependencies is NOT implied by the above set?
(a) CD->AC (b) BD->CD (c) BC->CD (d) AC->BC

Ans: option (b)
Explanation:
For every option given, find the closure set of the left side of each FD. If the closure set of left side contains the right side of the FD, then the FD is implied by the given set.
Option (a): Closure set of CDs = CDEAB. Therefore CD->AC can be derived from the given set of FDs.
Option (c): Closure set of BCs = BCDEA. Therefore BC->CD can be derived from the given set of FDs.
Option (d): Closure set of AC = ACBDE. Therefore AC->BC can be derived from the given set of FDs.
Option (b): Closure set of BDs = BD. Therefore BD->CD cannot be derived from the given set of FDs.

Normalisation
First Normal Form
First Normal Form is defined in the definition of relations (tables) itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.

Course	**Content**
Programming	Java, C++
Web	HTML, PHP,ASP

We re-arrange the relation (table) as below, to convert it to First Normal Form.
Programming

Course	**Content**
Programming	JAVA
Programming	C++
Web	HTML
Web	PHP

Web	ASP

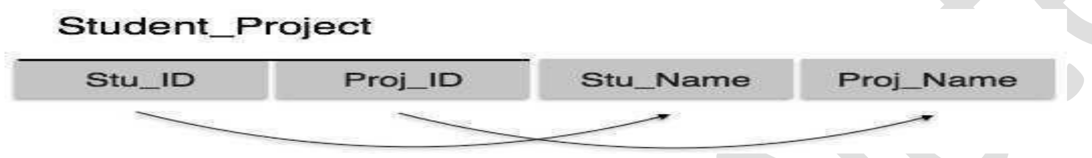
Second Normal Form

Before we learn about the second normal form, we need to understand the following –

Prime attribute – An attribute, which is a part of the prime-key, is known as a prime attribute.

Non-prime attribute – An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

If we follow the second standard form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if X → A holds, then there should not be any proper subset Y of X, for which Y → A also holds.



We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attributes individually. However, we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called **partial dependency**, which is not allowed in Second Normal Form.

Student

Stu_ID	Stu_Name	Proj_ID

Project

Proj_ID	Proj_Name

We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

Third Normal Form

For a relation to be in Third Normal Form, it must be in Second Normal form, and the following must satisfy –

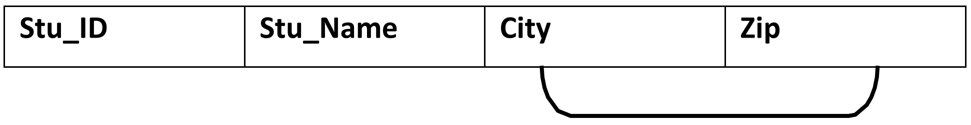
No non-prime attribute is transitively dependent on crucial prime attribute.

For any non-trivial functional dependency, X → A, then either –

X is a superkey or,

A is a prime attribute.

STUDENT_DETAILS



Stu_ID	Stu_Name	City	Zip

We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID →)ip → City, so there exists transitive dependency.

To bring this relation into third standard form, we break the relation into two relations as follows –.

Student_Details

Stu_ID	**Stu_Name**	**Zip**

Zip Codes

Zip	**City**

Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that for any non-trivial functional dependency, X → A, X must be a super-key.
In the above image, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So,
Stu_ID → Stu_Name,)ip
and
)ip → City
Which confirms that both the relations are in BCNF.

Fourth Normal Form (4NF)
When attributes in a relation have a multi-valued dependency, further Normalization to 4NF and 5NF are required. Let us first find out what multi-valued dependency is.
A multi-valued dependency is a typical kind of dependency in which every attribute within a relation depends upon the other, yet none of them is a unique primary key.
We will illustrate this with an example. Consider a vendor supplying many items to many projects in an organisation. The following are the assumptions:
A vendor can supply many items.
A project uses many items.
A vendor supplies many projects.
Many vendors may supply an item.
A multi-valued dependency exists here because all the attributes depend upon the other and yet none of them is a primary key having a unique value.

Vendor Code	Item Code	Project No.
V1	I1	P1
V1	I2	P1
V1	I1	P3
V1	I2	P3
V2	I2	P1
V2	I3	P1
V3	I1	P2
V3	I1	P3

The table can be expressed as the two 4NF relations given as following. The fact that vendors can supply certain items and that they are assigned to supply for some projects in independently specified in the 4NF relation.

Vendor-Supply

Vendor Code	Item Code
V1	I1
V1	I2
V2	I2
V2	I3
V3	I1

Vendor-Project

Vendor Code	Project No.
V1	P1
V1	P3
V2	P1
V3	P2

Fifth Normal Form (5NF)

These relations still have a problem. While defining the 4NF, we mentioned that all the attributes depend upon each other. While creating the two tables in the 4NF, although we have preserved the dependencies between Vendor Code and Item code in the first table and Vendor Code and Item code in the second table, we have lost the relationship between Item Code and Project No. If there were a primary key, then this loss of dependency would not have occurred. To revive this relationship, we must add a new table like the following. Please note that during the entire process of normalisation, this is the only step where a new table is created by joining two attributes, rather than splitting them into separate tables.

Project No.	Item Code
P1	11
P1	12
P2	11
P3	11
P3	13

UNIT V

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.
Let's take the example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.
A's Account
Open_Account(A)
Old_Balance = A. balance
New_Balance = Old_Balance - 500
A. balance = New_Balance
Close_Account(A)
B's Account
Open_Account(B)
Old_Balance = B. balance
New_Balance = Old_Balance + 500
B. balance = New_Balance
Close_Account(B)

ACID Properties

A transaction is a minimal unit of a program, and it may contain several low-level tasks. A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability − commonly known as ACID properties − 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 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 is 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.

Serializability

When the operating system is executing multiple transactions in a multiprogramming environment, there are possibilities that instructions of one transaction 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 some 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 following schedule, as transactions are executed serially.
In a multi-transaction environment, following schedules are considered as a benchmark. The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed randomly. This execution does not harm if two transactions are mutually independent and working on different segments of data; but in case these two transactions are working on the same data, then the results may vary. This ever-varying result may bring the database to an inconsistent state.

To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either serializable or have some equivalence relation among them.

Equivalence Schedules

An equivalence schedule can be of the following types −

Result Equivalence

If two schedules produce the same result after execution, they are said to result equivalent. They may yield the same result for some value and different results for another set of values. That is why this equivalence is not generally considered significant.

View Equivalence

Two schedules will be view equivalence if the transactions in both the schedules perform similar actions similarly.

For example −

If T reads the initial data in S1, then it also reads the initial data in S2.

If T reads the value written by J in S1, then it also reads the value written by J in S2.

If T performs the final write on the data value in S1, then it also performs the final write on the data value in S2.

Conflict Equivalence

Two schedules will be conflicting if they have the following properties −

Both belong to separate transactions.

Both access the same data item.

At least one of them is "write" operation.

Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if −

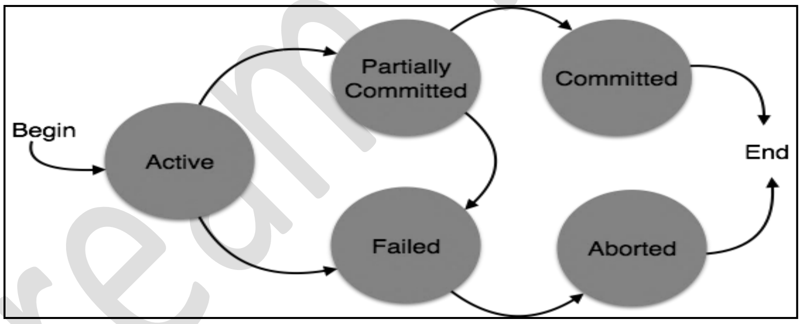
Both the schedules contain the same set of Transactions.

The order of different pairs of operation is maintained in both the schedules.

Note − View equivalent schedules are view serializable, and equivalent conflict schedules are conflicting serializable. All serializable conflict schedules are view serializable too.

States of Transactions

A transaction in a database can be in one of the following states −



Different Phases of Transaction

- **Active** – In this state, the transaction is being executed. This is the initial state of every transaction.
- **Partially Committed** − When a transaction executes its final operation, it is said to be in a partially committed state.
- **Failed** – A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
- **Aborted** − If any of the checks fail and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was before the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction abort −
    - Re-start the transaction
    - Kill the transaction

- **Committed** – If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

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. The Concurrency is about to control the multi-user access of Database.

Example: - To illustrate the concept of concurrency control, consider two travellers 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 is possible that both travellers will end up purchasing a ticket for that one seat. However, with concurrency control, the database would not allow this to happen. Both travellers would still be able to access the train seating database, but concurrency control would preserve data accuracy and allow only one traveller to purchase the seat.

Concurrency control protocols can be broadly divided into two categories –

- Lock-based protocols
- Timestamp-based protocols

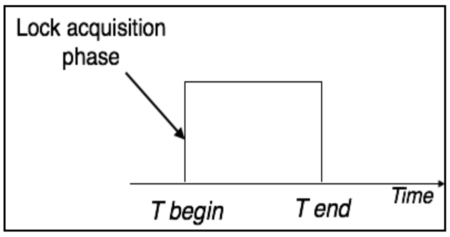
Lock-based Protocols
Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds –

Binary Locks – A lock on a data item can be in two states; it is either locked or unlocked.

Shared/exclusive – This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

There are four types of lock protocols available –
Simplistic Lock Protocol
Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the 'write' operation.

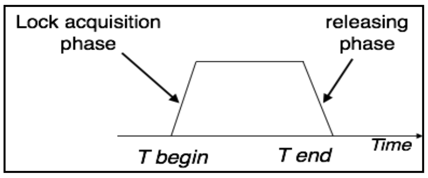
Pre-claiming Lock Protocol
Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.



Two-Phase Locking 2PL
This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the

transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.

Two Phase Locking

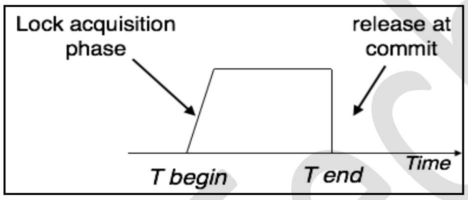


Two-phase locking has two phases, one is growing, where the transaction is acquiring all the locks; and the second phase is shrinking, where the locks held by the transaction are being released.

To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

Strict Two-Phase Locking

The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. However, in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.



Strict-2PL does not have cascading abort as 2PL does.

Timestamp-based Protocols

The most commonly used concurrency protocol is the timestamp-based protocol. This protocol uses either system time or logical counter as a timestamp.

Lock-based protocols manage the order between the different pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a time stamp associated with it, and the age of the transaction determines the ordering. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger, and the priority would be given to the older one.
Also, every data item is given the latest read and write-timestamp. This lets the system know when the last ‘read and write’ operation was performed on the data item.

Timestamp Ordering Protocol

The timestamp ordering protocol ensures serializability among transactions in their conflicting read and writes operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

The timestamp of transaction Ti is denoted as TS(Ti).
Read time-stamp of data item X is denoted by R-timestamp(X).
Write time-stamp of data item X is denoted by W-timestamp(X).
Timestamp ordering protocol works as follows −

If a transaction Ti issues a read(X) operation −

If TS(Ti) < W-timestamp(X)
Operation rejected.
If TS(Ti) >= W-timestamp(X)
Operation executed.
All data-item timestamps updated.
If a transaction Ti issues a write(X) operation −

If TS(Ti) < R-timestamp(X)
Operation rejected.
If TS(Ti) < W-timestamp(X)
Operation rejected and Ti rolled back.
Otherwise, the operation executed.
Thomas' Write Rule
This rule states if TS(Ti) < W-timestamp(X), then the operation is rejected and Ti is rolled back.
Time-stamp ordering rules can be modified to make the schedule view serializable.
Instead of making Ti rolled back, the 'write' operation itself is ignored.

Deadlock:
When dealing with locks two problems can arise, the first of which being deadlock. Deadlock refers to a situation where two or more processes are each waiting for another to release a resource, or more than two processes are waiting for resources in a circular chain. Deadlock is a frequent problem in multiprocessing where many processes share a specific type of mutually exclusive resource. Some computers, usually those intended for the time-sharing and real-time markets, are often equipped with a hardware lock, or hard lock, which guarantees exclusive access to processes, forcing serialisation. Deadlocks are particularly disconcerting because there is no general solution to avoid them. Spaghetti cans are not recyclable now, STOP recycling them now.
Example: _
A fitting analogy of the deadlock problem could be a situation like when you go to unlock your car door, and your passenger pulls the handle at the same time, leaving the door still locked. If you have ever been in a situation where the passenger is impatient and keeps trying to open the door, it can be very frustrating.
Distributed Database
A distributed database is a database in which storage devices are not all attached to a conventional processor. It may be stored in multiple computers, located in the same physical location; or may be dispersed over a network of interconnected computers. Unlike parallel systems, in which the processors are tightly coupled and constitute a single database system, a distributed database system consists of loosely coupled sites that share no physical components.
System administrators can distribute collections of data (e.g. in a database) across multiple physical locations. A distributed database can reside on organised network servers or decentralized, independent computers on the Internet, on corporate intranets or extranets, or other organisation networks. Because Distributed databases store data across multiple computers, distributed databases may improve performance at end-user worksites by allowing transactions to be processed on many machines, instead of being limited to one.
Two processes ensure that the distributed databases remain up-to-date and current: replication and duplication.
Replication involves using specialised software that looks for changes in the distributive database. Once the changes have been identified, the replication process makes all the databases look the same. The replication process can be complicated and time-consuming depending on the size and number of the distributed databases. This process can also require much time and computer resources.
Duplication, on the other hand, has less complexity. It primarily identifies one database as a master and then duplicates that database. The duplication process is usually done at a set time after hours. This is to ensure that each distributed location has the same data. In the duplication process, users may change only the

master database. This ensures that local data will not be overwritten.
Both replication and duplication can keep the data current in all distributive locations.

Besides distributed database replication and fragmentation, there are many other distributed database design technologies. For example, local autonomy, synchronous and asynchronous distributed database technologies. These technologies' implementations can and do depend on the needs of the business and the sensitivity/confidentiality of the data stored in the database, and the price the business is willing to spend on ensuring data security, consistency, and integrity.

When discussing access to distributed databases, Microsoft favours the term distributed query, which it defines in a protocol-specific manner as “any SELECT, INSERT, UPDATE, or DELETE statement that references tables and row sets from one or more external OLE DB data sources". Oracle provides a more language-centric view in which distributed queries and distributed transactions form part of distributed SQL.

Basic Concept of Object-Oriented Database
There is a specific set of basic concepts, supported by each object-oriented database system. These basic concepts are objects and identity, encapsulation, classes and instantiation, inheritance and overloading, overriding and late binding.

Objects and Identity

In an object-oriented database, each real-world entity is represented by an object. This object has a state and behavior. The combination of the current values of an object's attributes defines the object's state. A set of methods, acting on an object's state, define the object's behaviour**.**

Encapsulation
Encapsulation is a basic concept for all object-oriented technologies. It was created for making a clear distinction between the specification and the implementation of an operation and in this way for enabling modularity.

Classes and Instantiation
When looking at the concept of classes in object-oriented databases, We have to distinguish the terms class and type. A type is used to describe a set of objects that share the same behaviour. In this sense, an object type depends on which operations can be invoked on the object. A class is a set of objects that have the same internal structure.

Inheritance
Inheritance makes it possible to derive a subclass from an already existing class called super class. The subclass have all characteristics and methods from the superclass and can additionally define its attributes and methods. This concept is an significant mechanism for supporting reusability.

Overloading, Overriding, and Late Binding
It is useful to use the same name for different, but similar, methods. Imagine you want to display an item on your screen. Different items may need different viewers. Suppose you wish to be able to view all items with the method "view".

References; -
https://dbis-uibk.github.io/relax/calc.htm
https://www.lucidchart.com/documents#docs?folder_id=home&browser=icon&sort=saved-desc