**RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL**

**New Scheme Based On AICTE Flexible Curricula**

**CSE-Artificial Intelligence and Machine Learning/ Artificial Intelligence and Machine Learning, V-Semester**

**AL-502 Database Management Systems**

**COURSE OBJECTIVES:** The objective of this course is to enable students in developing a high level understanding of the concepts of Database management systems in contrast with traditional data management systems with emphasis on skills to apply these concepts in building, maintaining and retrieving data from these DBMS.

**COURSE OUTCOMES:**

**After completing the course student should be able to:**

1. Describe design of a database at various levels and compare andcontrast traditional data processing with DBMS.
2. Design a database using Entity Relationship diagram and other design techniques.
3. Apply fundamentals of relational model to model and implement a sample Database Management System for a given domain.
4. Evaluate and optimize queries and apply concepts of transaction management.



**COURSE CONTENTS:**

**UNIT I**:DBMS Concepts and architecture Introduction, Database approach v/s Traditional file accessing approach, Advantages of database systems, Data models, Schemas and instances, Data independence, Data Base Language and interfaces, Overall Database Structure, Functions of DBA and designer, ER data model: Entitles and attributes, Entity types, Defining the E-R diagram, Concept of Generalization, Aggregation and Specialization. Transforming ER diagram into the tables. Various other data models object oriented data Model, Network data model, and Relational data model, Comparison between the three typesof models.Storage structures: Secondary Storage Devices, Hashing & Indexing structures: Single level & multilevel indices.

**UNIT II:**Relational Data models: Domains, Tuples, Attributes, Relations, Characteristics of relations, Keys, Key attributes of relation, Relational database, Schemas, Integrity constraints. Referential integrity, Intension and Extension, Relational Query languages: SQL-DDL, DML, integrity con straints, Complex queries, various joins, indexing, triggers, assertions , Relational algebra and relational calculus, Relational algebra operations like select, Project ,Join, Division, outer union. Types of relational calculus i.e. Tuple oriented and domain oriented relational calculus and its operations.

**UNIT III:** Data Base Design: Introduction to normalization, Normal forms- 1NF, 2NF, 3NF

and BCNF, Functional dependency,Decomposition, Dependency preservation and lossless join, problems with null valued and dangling tuples, multivalued dependencies. Query Optimization: Introduction, steps of optimization, various algorithms to implement select, project and join operations of relational algebra, optimization methods: heuristic based, cost estimation based.

**UNIT IV:**Transaction Processing Concepts: -Transaction System, Testing of Serializability, Serializability of schedules, conflict & view serializable schedule, recoverability, Recoveryfrom transaction failures. Log based recovery. Checkpoints deadlock handling. Concurrency Control Techniques: Concurrency Control, locking Techniques for concurrency control, timestamping protocols for concurrency control, validation based protocol, multiple granularity.Multi version schemes, Recovery with concurrent transaction. Introduction to Distributeddatabases, data mining, data warehousing, Object Technology and DBMS, Comparativestudy of OODBMS Vs DBMS . Temporal, Deductive, Multimedia, Web & Mobile database.

**UNIT V:**Case Study of Relational Database Management Systems through Oracle/PostgreSQL /MySQL: Architecture, physical files, memory structures, background process. Data dictionary, dynamic performance view. Security, role management, privilege management, profiles, invoker defined security model. SQL queries, Hierarchical quires, inline queries, flashback queries. Introduction of ANSI SQL, Cursor management: nested and parameterized cursors. Stored procedures, usage of parameters in procedures. User defined functions their limitations. Triggers, mutating errors, instead of triggers.



TEXT BOOKS RECOMMENDED:

1. Korth H.F. &Silberschatz A., Sudarshan, “Database Systems”, McGraw-Hill
2. Chris J. Date, with Hugh Darwin, Addison-Wesley, “A Guide to SQL Standard”.
3. Elmasri R., Navathe S.B., “Fundamentals of Database Systems”, Pearson.

REFERENCE BOOKS:

1. Rob, “ Database System:Design Implementation& Management”, Cengage Learning.
2. AtulKahate , “Introduction to Database Management System”, Pearson Educations
3. Oracle 9i Database Administration Fundamental-I, Volume I, Oracle Press,TMH.
4. Paneerselvam,”Database Management System”, PHI Learning

# Subject Notes [DBMS CS 502]

## INTRODUCTION

**Data:-**Data is a plural of datum, which is originally a Latin noun meaning "something given." Today, data is used in English both as a plural noun meaning "facts or pieces of information" and as a singular mass noun meaning "information".

Data can be defined as a representation of facts, concepts or instructions in a formalized manner which should be suitable for communication, interpretation, or processing by human or electronic machine.

Data is represented with the help of characters like alphabets (A-Z,a-z), digits (0-9) or special characters(+,-, /,*, <,>, = etc.).

Types of Data: Text, Numbers & Multimedia

**Information: -**

Information is organized or classified data which has some meaningful values for the receiver.

Information is the processed data on which decisions and actions are based.

For the decision to be meaningful, the processed data must qualify for the following characteristics:

- **Timely -** Information should be available when required.
- **Accuracy -** Information should be accurate.
- **Completeness -** Information should be complete.

**File System**

Before DBMS invented, information was stored in file processing system. A file processing system is a collection of files and programs that access/modify these files. Typically, new files and programs are added over time (by different programmers) as new information needs to be stored and new ways to access information are needed.

Problems with file processing systems:



**File System Verses Database**

Most explicit and major disadvantages of file system when compared to database management system are as follows:

**Data Redundancy**- The files are formed in the file system as and when required by an enterprise over its progress path. So, in that case the repetition of information about an entity cannot be avoided.E.g. The addresses of customers will be present in the file keeping information about customers holding savings account and also the address of the customers will be present in file maintaining the current account. Even when same customers have a saving account and current account his address will be present at two places.

**Data Inconsistency**: Data redundancy leads to bigger problem than just wasting the storage i.e. it may lead to inconsistent data. Same data which has been frequented at several places may not match after it has been updated at some places.

For example: Suppose the customer requests to change the address for his account in the Bank and the Program is executed to update the saving bank account file only but his current bank account file is not updated. Afterwards the addresses of the same customer present in saving bank account file and current bank account file will not match.

Moreover, there will be no way to find out which address is latest out of these two.

**Difficulty in Accessing Data**: For producing ad hoc reports the programs will not already be present and only options present will to write a new program to generate requested report or to work manually. This is going to take impractical time and will be more expensive.

For example: Suppose all of sudden the administrator gets a request to generate a list of all the customers holding the saving banks account who lives in particular locality of the city. Administrator will not have any program already written to generate that list but say he has a program which can generate a list of all the customers holding the savings account. Then he can either provide the information by going thru the list

manually to select the customers living in the particular locality or he can write a new program to generate the new list. Both of these ways will take large time which would generally be impractical.

**Data Isolation**: Since the data files are formed at different times and supposedly by different person, the structures of different files generally will not match. The data will be scattered in different files for a particular entity. So, it will be difficult to get appropriate data.

For example: Suppose the Address in Saving Account file have fields: Add line1, add line2, City, State, Pin while the fields in address of Current account are: House No., Street No., Locality, City, State, and Pin. Administrator is asked to provide the list of customers living in a particular locality. Providing consolidated list of all the customers will require looking in both files. But they both have different way of storing the address.

Writing a program to generate such a list will be difficult.

**Integrity Problems**: All the consistency constraints have to be applied to database through appropriate drafts in the coded programs. This is very difficult when number such constraint is very large.

For example: An account should not have balance less than Rs. 500. To enforce this constraint appropriate check should be added in the program which add a record and the program which withdraw from an account. Suppose later on this amount limit is increased then all those checks should be updated to avoid inconsistency. These time to time changes in the programs will be great headache for the administrator.

Security and access control: Database should be protected from unauthorized users.Every user should not be allowed to access every data. Since application programs are added to the system. For example: The Payroll Personnel in a bank should not be allowed to access accounts information of the customers.

**Concurrency Problems**: When more than one user is permitted to process the database.If in that environment two or more users try to update a shared data element at about the same time then it may result into inconsistent data. For example, Suppose Balance of an account is Rs. 500. And User A and B try to withdraw Rs. 100 and Rs. 50 respectively at almost the same time using the Update process.

Update:

1. Read the balance amount.
2. Subtract the withdrawn amount from balance.
3. Write updated Balance value.



Suppose A performs Step 1 and 2 on the balance amount i.e. it reads 500 and subtracts 100 from it. But at the same time B withdraws Rs 50 and he performs the Update process and he also reads the balance as 500 subtract 50 and writes back 450. User A will also write his updated Balance amount as 400. They may update the Balance value in any order depending on various reasons concerning to system being used by both of the users. So finally, the balance will be either equal to 400 or 450. Both of these values are wrong for the updated balance and so now the balance amount is having inconsistent value forever.

**Advantages of database systems**

- **Data Independence**: Application programs should not, if possible, be showing to details of data representation and storage, The DBMS provides an abstract view of the data that hides such details.
- **Efficient Data Access:** A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.
- **Data Integrity and Security:** If data is always retrieved through the DBMS, the DBMS can implement integrity constraints. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded. Also, it can enforce access controls that govern what data is visible to different classes of users.
- **Data Administration:** When several users share the data, centralizing the administration of data can offer significant improvements. Experienced professionals, who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and for fine-tuning the storage of the data to make retrieval efficient.

- **Concurrent Access and Crash Recovery:** A DBMS schedules simultaneous accesses to the data in such a manner that users can think of the data as being accessed by only one user at a time. Further, the DBMS protects users from the effects of system failures.
- **Reduced Application Development Time:** Clearly, the DBMS supports important functions that are common to many applications accessing data in the DBMS. This, in conjunction with the high-level interface to the data, facilitates quick application development. DBMS applications are also likely to be more robust than similar stand-alone applications because many important tasks are handled by the DBMS (and do not have to be debugged and tested in the application).

**What Is Database:**

A database consists of an organized collection of interrelated data for one or more uses, typically in digital form.Examples of databases could be: Database for Educational Institute or a Bank, Library, Railway Reservation system etc.

- Consists of two things- a Database and a set of programs.
- Database is a very large, integrated collection of data.
- The set of programs are used to Access and Process the database.

So, DBMS can be defined as the software package designed to store and manage or process the database.

**Management of data involves**

- Definition of structures for the storage of information
- Methods to manipulate information
- Safety of the information stored despite system crashes.

Database model's real-world enterprise by entities and relationships.

Entities (e.g., students, courses, class, subject)



**CHARACTERISTICS OF DATABASE APPROACH**

A number of characteristics distinguish the database method from the much older approach of programming with files. In traditional file processing, each user defines and implements the files needed for a precise software application as part of programming the application.

For example, one user, the grade reporting office, may keep files on students and their grades. Programs to print a student's transcript and to enter new grades are implemented as part of the application.

A second user, the accounting office, may keep track of students' fees and their payments. Although both users are interested in data about students, each user maintains separate files—and programs to manipulate these files—because each requires some data not available from the other user's files. This redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date data. In the database approach, a single repository maintains data that is defined once and then accessed by various users. In file systems, each application is free to name data elements independently. In contrast, in a database, the names or labels of data are defined once, and used repeatedly by queries, transactions, and applications.

The main characteristics of the database approach versus the file-processing approach are the following:

- Self-describing nature of a database system
- Insulation between programs and data, and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing.

**Data models**

A data model—a collection of concepts that can be used to describe the structure of a database—provides the essential means to achieve the abstraction. By structure of a database we mean the data types, relationships,

and constraints that apply to the data. Most data models also contain a set of basic operations for specifying retrievals and updates on the database.

In addition to the basic operations provided by the data model, it is becoming more common to include concepts in the data model to specify the dynamic aspect or behavior of a database application. This allows the database designer to specify a set of valid user-defined operations that are allowed on the database objects.

Data models define how the logical structure of a database is modeled. Data Models are fundamental entities to introduce abstraction in a DBMS. Data models define how data is connected to each other and how they are processed and stored inside the system.

**Types of Database Users:**

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

1. **Application programmers** - interact with system through DML calls.
2. **Sophisticated users** - form requests in a database query language.
3. **Specialized users** - write specialized database applications that do not fit into the traditional data processing framework.
4. **Naive users** - invoke one of the permanent application programs that have been written previously.

**Three Level Architecture of DBMS-**

An early proposal for a standard terminology and general architecture database a system was produced in 1971 by the DBTG (Data Base Task Group) appointed by the Conference on data Systems and Languages. The DBTG recognized the need for a two-level approach with a system view called the schema and user view called subschema. The American National Standard Institute terminology and architecture in 1975.ANSI-SPARC recognized the need for a three-level approach with a system catalog.

**There are following three levels or layers of DBMS architecture:**

1. External Level
2. Conceptual Level
3. Internal Level

**1. External Level**: - External Level is described by a schema i.e. it consists of definition of logical records and relationship in the external view. It also contains the method of deriving the objects in the external view from the objects in the conceptual view.

**2. Conceptual Level**: - Conceptual Level represents the entire database. Conceptual schema describes the records and relationship included in the Conceptual view. It also contains the method of deriving the objects in the conceptual view from the objects in the internal view.

**3. Internal Level: -** Internal level indicates hoe the data will be stored and described the data structures and access method to be used by the database. It contains the definition of stored record and method of representing the data fields and access aid used.

A mapping between external and conceptual views gives the correspondence among the records and relationship of the conceptual and external view. The external view is the abstraction of conceptual view which in turns is the abstraction of internal view. It describes the contents of the database as perceived by the user or application program of that view.

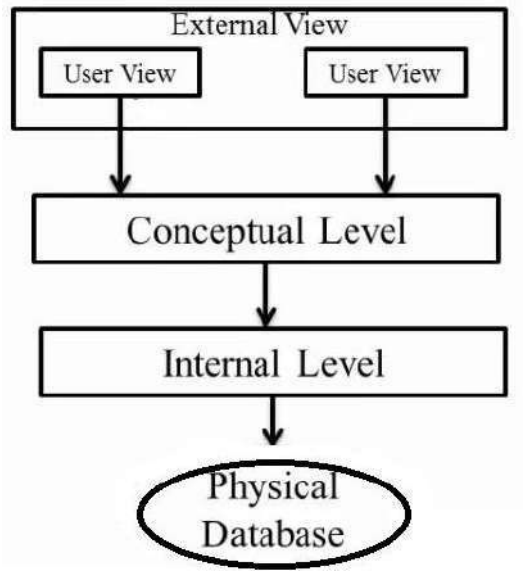



**Fig-1.1**

**Difference between Two Tier and Three Tier Architecture**

| Sr. No | Two-tier Architecture | Three -tier Architecture |
|---|---|---|
| 1 | Client -Server Architecture | Web -based application |
| 2 | Client will hit request directly to server and client will get response directly from server | Here in between client and server middle ware will be there, if client hits a request it will go to the middle ware and middle ware will send to server and vice versa. |
| 3 | 2-tier means 1) Design layer 2) Data layer | 3-tier means 1) Design layer 2) Business layer or Logic layer 3) Data layer |



**Three-Tier Architecture:**

**Three-tier architecture** typically comprises a presentation tier, a business or data access tier, and a data tier. Three layers in the three-tier architecture are as follows:

**1) Client layer**
**2) Business layer**
**3) Data layer**

**1) Client layer:**

It is also called as Presentation *layer* which contains UI part of our application. This layer is used for the design purpose where data is presented to the user or input is taken from the user.

Example-Designing registration form which contains text box, label, button etc.

**2) Business layer:**

In this layer, all business logic likes validation of data, calculations, data insertion etc. This is an interface between Client layer and Data Access Layer. This layer is also called the intermediary layer helps to make communication faster between client and data layer.

**3) Data layer:**

In this layer, actual database is coming in the picture. Data Access Layer contains methods to join with database and to perform insert, update, delete, get data from database based on our input data.

**Advantages**

1. High performance, lightweight persistent objects

2. Scalability – Each tier can scale horizontally
3. Performance – Because the Presentation tier can cache requests, network utilization is minimized, and the load is reduced on the Application and Data tiers.
4. High degree of flexibility in deployment platform and configuration
5. Better Re-use
6. Improve Data Integrity
7. Improved Security – Client is not direct access to database.
8. Easy to maintain and modification is bit easy, won't affect other modules
9. In three tier architecture application performance is good.

**Disadvantages**

1. Increase Complexity/Effort

**RDBMS-It stands for Relational Database Management System. It organizes data into related rows and columns.**

**Features:**

It stores data in tables.
Tables have rows and column.
These tables are created using SQL.

**Difference between DBMS and RDBMS**

| No. | DBMS | RDBMS |
|---|---|---|
| 1) | DBMS applications store **data as file**. | RDBMS applications store **data in a tabular form**. |
| 2) | In DBMS, data is generally stored in either a hierarchical form or a navigational form. | In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables. |
| 3) | **Normalization is not** present in DBMS. | **Normalization is** present in RDBMS. |
| 4) | DBMS does **not apply any security** with regards to data manipulation. | RDBMS **defines the integrity constraint** for the purpose of ACID (Atomicity, Consistency, Isolation and Durability) property. |
| 5) | DBMS uses file system to store data, so there will be **no relation between the tables**. | In RDBMS, data values are stored in the form of tables, so a **relationship** between these data values will be stored in the form of a table as well. |
| 6) | DBMS has to provide some uniform methods to access the stored information. | RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information. |
| 7) | DBMS **does not support distributed database**. | RDBMS **supports distributed database**. |
| 8) | DBMS is meant to be for small organization and **deal with small data**. It supports **single user**. | RDBMS is designed to **handle large amount of data**. It supports **multiple users**. |
| 9) | Examples of DBMS are file systems, **xml** etc. | Example of RDBMS are **MySQL**, **postgrad**, **sql server**, **oracle** etc. |

**Types of data models are:**

- Entity relationship model
- Relational model
- Hierarchical model
- Network model
- Object oriented model
- Object relational model

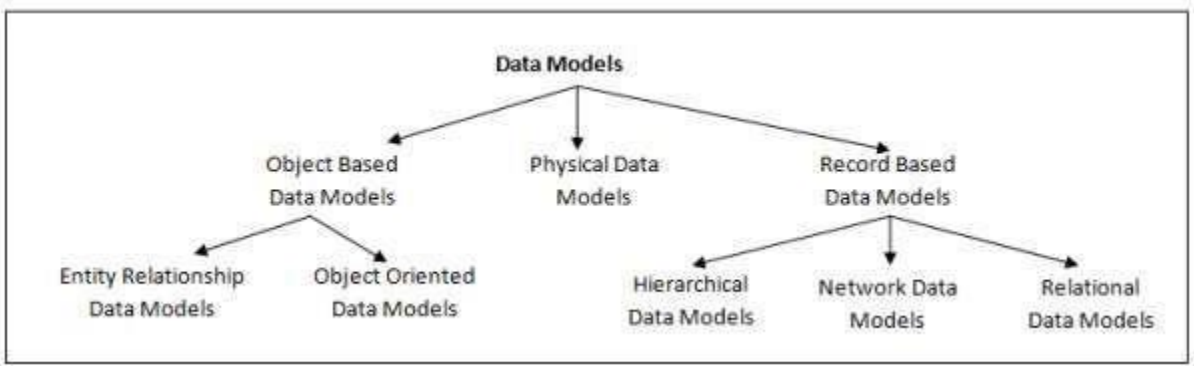



**Fig. 1.2**

**Entity relationship model**

**Entity**

An entity can be real world object, either animate or inanimate, that can be simplyrecognizable. For example, in a school database, students, teachers, classes and courses offered can be considered as entities. Entities are represented by means of rectangles.



**Relationship**

A relationship is an association among several entities. For example, an employee works at a department, a student enrolls in a course. Here, **works at** and **enrolls** are called relationship. Relationships are represented by diamond-shaped box.

**Attributes**

Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).

- **Strong Entity**: Entities having its particular attribute as primary keys are called strong entity. For example, STUDENT has STUDENT_ID as primary key. Hence it is a strong entity.
- **Weak Entity**: Entities which cannot form their own attribute as primary key are known weak entities. These entities will derive their primary keys from the combination of its attribute and primary key from its mapping entity.
- **Simple Attribute**

These kinds of attributes have values which cannot be divided further. For example, STUDENT_ID attribute which cannot be divided further. Passport Number is unique value and it cannot be divided.

- **Composite Attribute**

This kind of attribute can be divided further to more than one simple attribute. For example, address of a person. Here address can be further divided as Door#, street, city, state and pin which are simple attributes.

- **Derived Attribute**

Derived attributes are the one whose value can be obtained from other attributes of entities in the database. For example, Age of a person can be obtained from date of birth and current date. Average salary, annual salary, total marks of a student etc. are few examples of derived attribute.

- **Stored Attribute**

The attribute which gives the value to get the derived attribute are called Stored Attribute. In example above, age is derived using Date of Birth. Hence Date of Birth is a stored attribute.

- **Single Valued Attribute**

These attributes will have only one value. For example, EMPLOYEE_ID, passport#, driving license#, SSN etc have only single value for a person.

- **Multi-Valued Attribute**

These attributes can have more than one value at any point of time. Manager can have more than one employee working for him, a person can have more than one email address, and more than one house etc is the examples.

- **Simple Single Valued Attribute**

This is the combination of above four types of attributes. An attribute can have single value at any point of time, which cannot be divided further. For example, EMPLOYEE_ID – it is single value as well as it cannot be divided further.

- **Simple Multi-Valued Attribute**

Phone number of a person, which is simple as well as he can have multiple phone numbers is an example of this attribute.

- **Composite Single Valued Attribute**

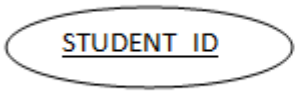
Date of Birth can be a composite single valued attribute. Any person can have only one DOB and it can be further divided into date, month and year attributes.

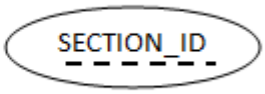
- **Composite Multi-Valued Attribute**

Shop address which is located two different locations can be considered as example of this attribute.

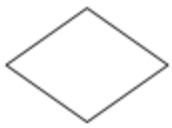
- **Descriptive Attribute**

Attributes of the relationship is called descriptive attribute. For example, employee works for department. Here 'works for' is the relation between employee and department entities. The relation 'works for' can have attribute DATE_OF_JOIN which is a descriptive attribute.




**Relationship**: A diamond shape is used to show the relationship between the entities. A mapping with weak entity is shown using double diamond. Relationship name will be written inside them.

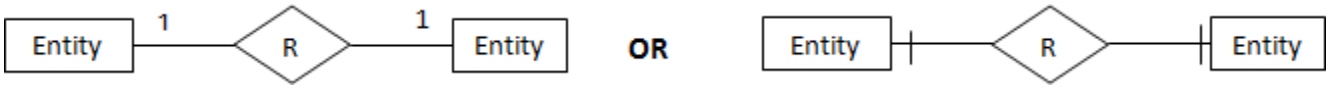


Normal Relations

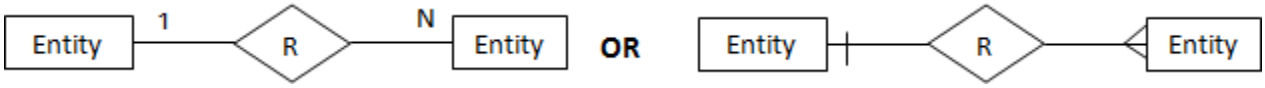
Identifying or weak Relations

**Cardinality of Relationship**: Different developers use different notation to represent the cardinality of the relationship. Not only for cardinality, but for other objects in ER diagram will have slightly different notations. But main difference is noticed in the cardinality. For not to get confused with many, let us see two types of notations for each.

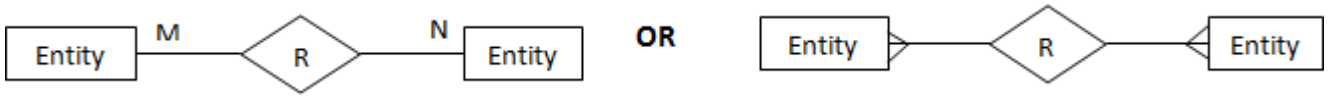
**One-to-one relation: -** A one-to-one relationship is represented by adding '1' near the entities on the line joining the relation. In another type of notation one dash is added to the relationship line at both ends.



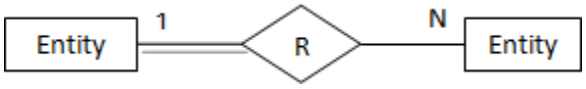
**One-to-Many relation**: A one-to-many relationship is represented by adding '1' near the entity at left hand side of relation and 'N' is written near the entity at right side. Other type of notation will have dash at LHS of relation and three arrow kinds of lines at the RHS of relation as shown below.




**Many-to-Many relation**: A one-to-many relationship is represented by adding 'M' near the entity at left hand side of relation and 'N' is written near the entity at right side. Other type of notation will have three arrow kinds of lines at both sides of relation as shown below.




**Participation Constraints**: Total participation constraints are shown by double lines and partial participations are shown as single line.




**Data-model**

In the below diagram, Entities or real-world objects are represented in a rectangular box. Their attributes are represented in ovals. Primary keys of entities are underlined. All the entities are mapped using diamonds. This is one of the methods of representing ER model. There are many different forms of representation. More details of this model are described in ER data model article.

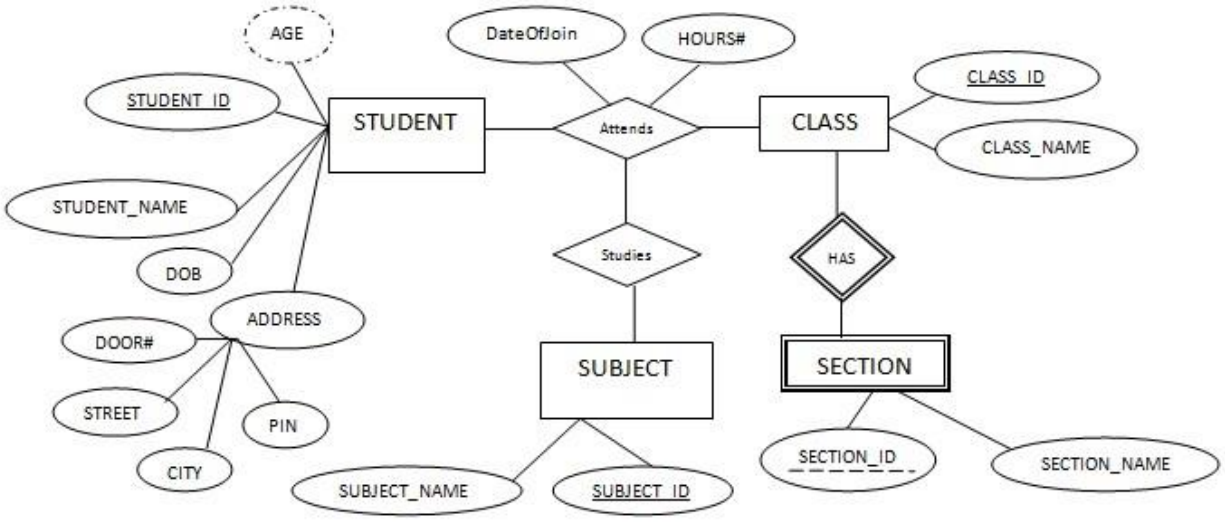



**Fig 1.3**

Basically, ER model is a graphical representation of real world objects with their attributes and relationship. It makes the system easily understandable. This model is considered as a top down approach of designing a requirement.

**Advantages:**

- It makes the requirement simple and easily reasonable by representing simple diagrams.

- One can covert ER diagrams into record based data model easily.
- Easy to understand ER diagrams

**Disadvantages:**

- No standard notations are available for ER diagram. There is great flexibility in the notation. It's all depends upon the designer, how he draws it.
- It is meant for high level designs. We cannot simplify for low level design like coding.

**Object Oriented Data Model**

This data model is another method of representing real world objects. It considers each object in the world as objects and isolates it from each other. It groups its related functionalities together and allows inheriting i ts functionality to other related sub-groups.

Let us consider an Employee database to understand this model better. In this, we have different types of employees – Engineer, Accountant, Manager, Clark. But all these employees belong to Person group. Person can have different attributes like name, address, age and phone. What do we do if we want to get a person's address and phone number? We write two separate procedure sp_getAddress and sp_getPhone.

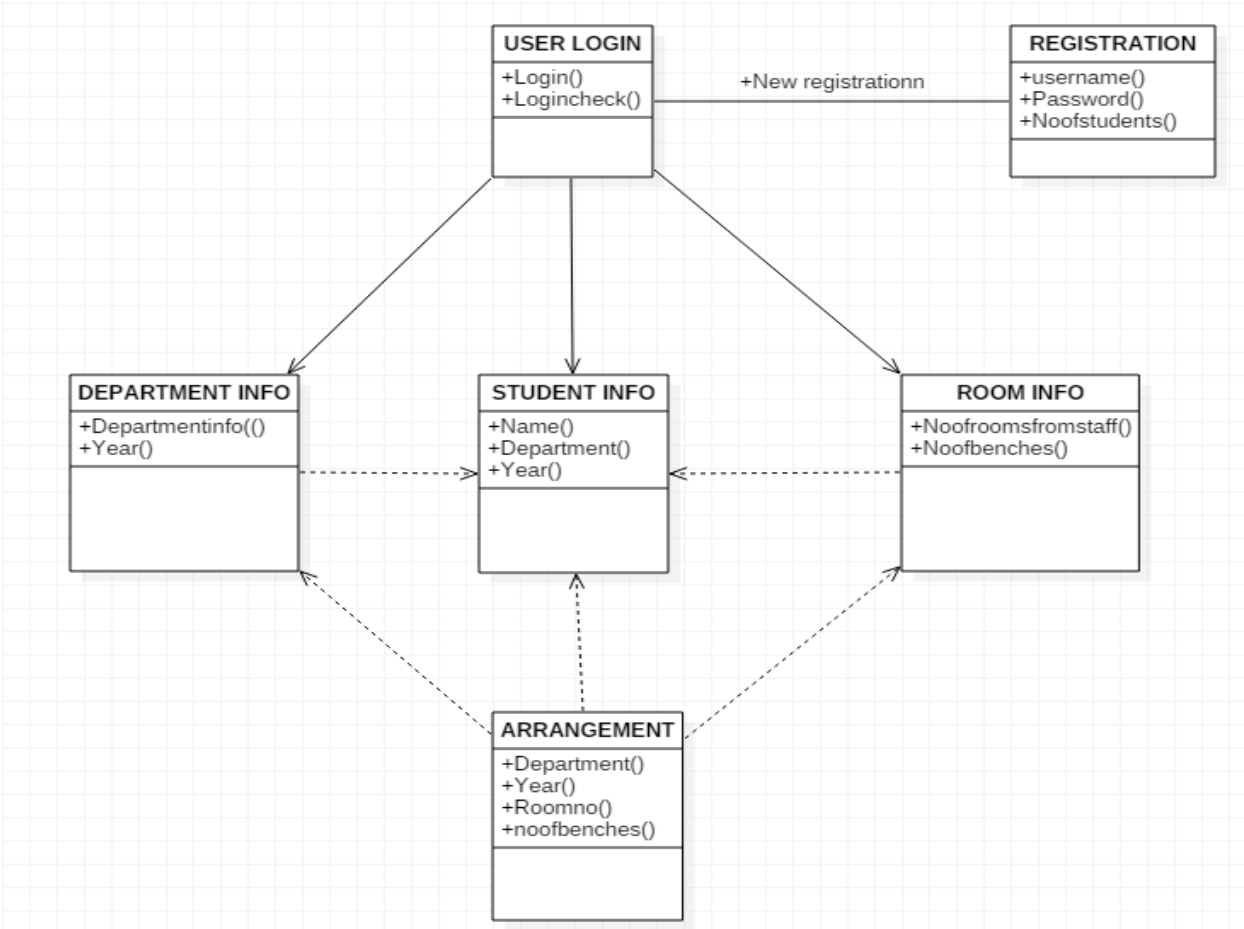



**Fig 1.4**

**Advantages:**

- We can re-use the attributes and functionalities. It reduces the cost of maintaining the same data at multiple times. Also, these information's are encapsulated and, there is no fear being changed by other objects. If we need any new feature we can easily add new class inherited from parent class and adds new features. Hence it reduces the overhead and maintenance costs.
- Because of the above feature, it becomes more flexible in the case of any changes.
- Codes are re-used because of inheritance.
- Since each class binds its attributes and its functionality, it is same as the real-world object. We can see each object as a real entity. Hence it is more understandable.

**Disadvantages:**

- It is not widely developed and complete to use it in the database systems. Hence it is not accepted by the users.
- It is amethod for solving the requirement. It is not a technology. Hence it fails to put it in the database management systems.

Imagine we have to create a database for a company. What are the entities involved in it? Company, its department, its supplier, its employees, different projects of the company etc are the different entities we need to take care of. If we observe each of the entity they have parent –child relationship. We can design them like we do ancestral hierarchy. In our case, Company is the parent and rests of them are its children. Department has employees and project as its children and so on. This type of data modeling is called hierarchical data model.

In this data model, the entities are represented in a hierarchical fashion. Here we identify a parent entity, and its child entity. Again, we drill down to identify next level of child entity and so on. This model can be imagined as folders inside a folder!

In our example above, it is diagrammatically represented as below:

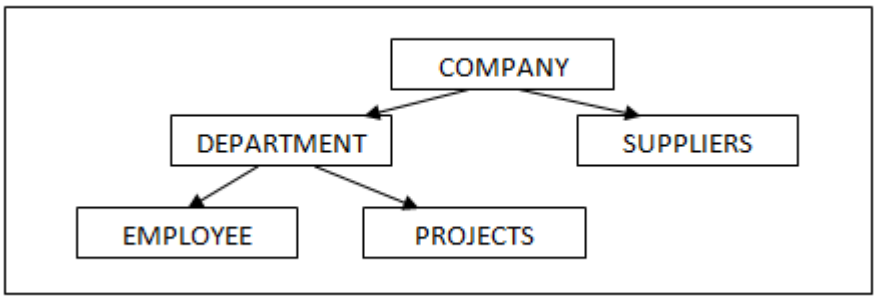





**Fig 1.5**

It can also be imagined as root like structure. This model will have only one main root. It then branches into sub-roots, each of which will branch again. This type of relationship is best defined for 1 :N type of relationships. E.g.; One company has multiple departments (1:N), one company has multiple suppliers (1:N),one department has multiple employees (1:N), each department has multiple projects(1:N) . If we have M:N relationships, then we have to duplicate the entities and show it in the diagram. For example, if a project in the company involves multiple departments, then our hierarchical representation changes as below:

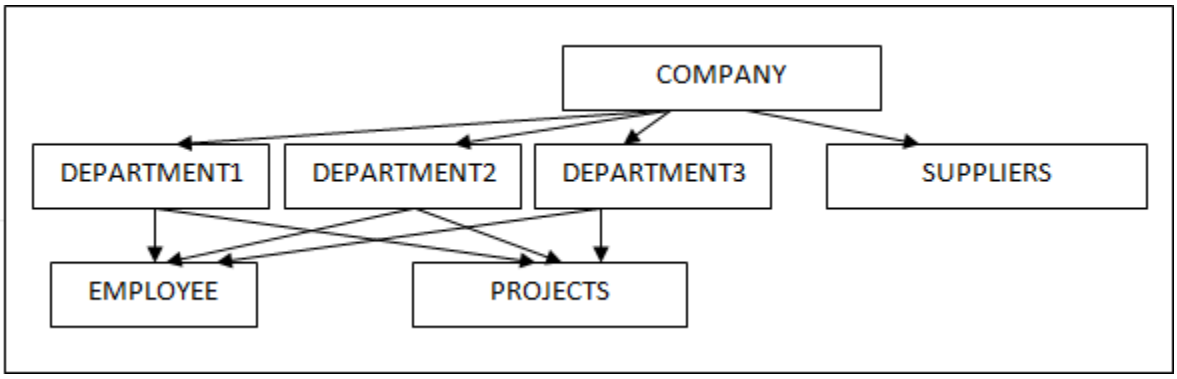



**Fig 1.6**

**Advantages**

It helps to address the issues of flat file data storage. In flat files, data will be scattered and there will not be in a proper structure. This model groups the related data into tables and defines the relationship between the tables, which is not addressed in flat files.

**Disadvantages**

• Redundancy: - When data is stored in a flat file, there is chance of repetition of same data multiple times and any changes required for the data will need to change in all the places in the flat file. Missing to update at any one place will cause incorrect data. This kind redundancy is solved by hierarchical model to some extent. Since records are grouped under related table, it solves the flat file redundancy issue. But look at the many to many relationship examples given above. In such case, we have to store same project information for more than one department. This is duplication of data and hence a redundancy. So, this model does not reduce the redundancy issue to a significant level.

• As we have seen above, it fails to handle many to many relationships competently. Results in redundant and having confusion. It can handle only parent-child kind of relationship.

• If we need to fetch any data in this model, we have to start from the root of the model and traverse through its child till we get the result. In order to perform the traversing, either we should know well in advance the layout of model or we should be very good programmer. Hence fetching through this model becomes bit difficult.

• Imagine company has got some new project details, but it did not assign it to any department yet. In this case, we cannot store project information in the PROJECT table, till company assigns it to some department. That means, in order to enter any child information, its parent information should be already known / entered.

**Network Data Models**

This is the improved version of hierarchical data model. It is designed to address the drawbacks of the hierarchical model. It helps to address M: N relationship. This data model is also represented as hierarchical, but this model will not have single parent relationship. Any child in the tree can have multiple parents here.

Revisit our company example: A company has different projects and departments in the company own those projects. Even suppliers of the company give input for the project. Here Project has multiple parents and each department and supplier have multiple projects. This is represented as shown below. Basically, it forms a network like structure between the entities, hence the name.

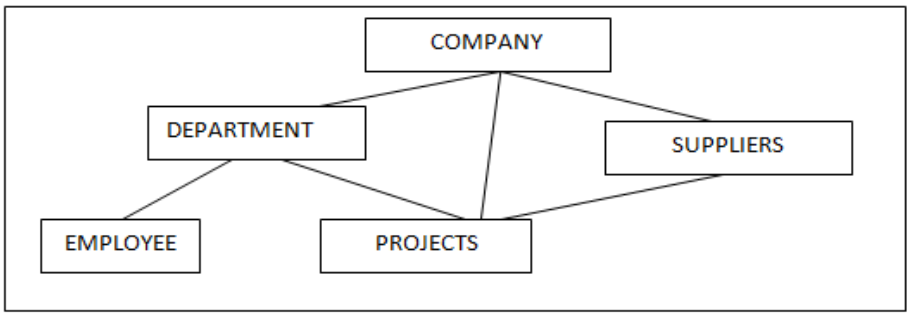



**Fig 1.7**

One more example:

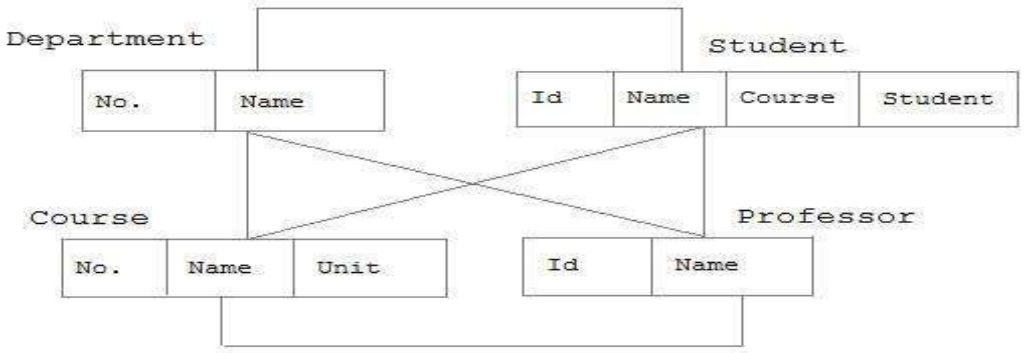



**Fig 1.8**

**Advantages**

- Accessing the records in the tables is easy since it addresses many to many relationships. Because of this kind of relationship, any records can be easily pulled by using any tables. For example, if we want to know the department of project X and if we know SUPPLIER table, we can pull this information. i.e.; SUPPLIER has the information about project X which includes the departments involved in the projects too. Hence makes the accessibility to any data easier, and even any complex data can be retrieved easily and quickly.
- Because of the same feature as per first point, one can easily navigate among the tables and get any data.
- It is designed based on database standards – ANSI/SP ARC.

**Disadvantages**

- If there is any requirement for the changes to the entities, it requires entire changes to the database. There is no independence between any objects. Hence any changes to the any of the object will need changes to the whole model. Hence difficult to manage.
- It would be little difficult to design the relationship between the entities, since all the entities are related in some way. It requires thorough practice and knowledge about the designing.

**Relational Data Models**

This model is designed to overcome the drawbacks of hierarchical and network models. It is designed completely different from those two models. Those models define how they are structured in the database physically and how they are inter-related. But in the relational model, we are least concerned about how they are structured. It purely based on how **the records in each table are related**. It purely isolates physical structure from the logical structure. Logical structure is defining records are grouped and distributed.

Let us try to understand it by an example. Let us consider department and employee from our previous examples above. In this model, we look at employee with its data. When we say an employee what all comes into our mind? His employee id, name, address, age, salary, department that he is working etc. are attributes of employee. That means these details about the employee forms columns in employee table and value set of each employee for these attribute forms a row/record for an employee. Similarly, department has its id, name.

Now, in the employee table, we have column which uniquely identifies each employee – that is employee Id column. This column has unique value and we are able to differentiate each employee from each other by using this column. Such column is called as primary key of the table. Similarly, department table has DEPT_ID as primary key. In the employee table, instead of storing whole information about his department, we have DEPT_ID from department table stored. i.e.; by using the data from the department table, we have established the relation between employee and department tables.

**EMPLOYEE**

| EMP_ID | EMP_NAME | ADDRESS | DEPT_ID |
|---|---|---|---|
| 100 | Joseph | Clinton Town | 10 |
| 101 | Rose | Fraser Town | 20 |
| 102 | Mathew | Lakeside Village | 10 |
| 103 | Stewart | Troy | 30 |
| 104 | William | Holland | 30 |

**DEPARTMENT**

| DEPT_ID | DEPT_NAME |
|---|---|
| 10 | Accounting |
| 20 | Quality |
| 30 | Design |
| | |

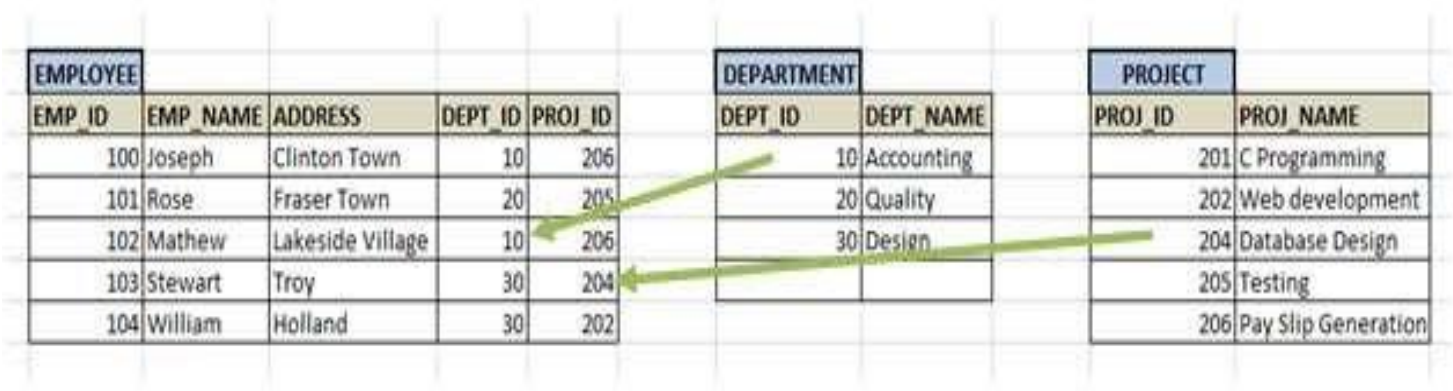
**Fig 1.9**

A relational data model revolves around 5 important rules.

1. Order of rows / records in the table is not important. For example, displaying the records for Joseph is independent of displaying the records for Rose or Mathew in Employee table. It does not change the meaning or level of them. Each record in the table is independent of other. Similarly, order of columns in the table is not important. That means, the value in each column for a record is independent of other. For example, representing DEPT_ID at the end or at the beginning in the employee table does not have any affect.

2. Each record in the table is unique. That is there is no duplicate record exists in the table. This is achieved by the use of primary key or unique constraint.
3. Each column/attribute will have single value in a row. For example, in Department table, DEPT_NAME column cannot have 'Accounting' and 'Quality' together in a single cell. Both has to be in two different rows as shown above.
4. All attributes should be from same domain. That means each column should have meaningful value. For example, Age column cannot have dates in it. It should contain only valid numbers to represent individual's age. Similarly, name columns should have valid names, Date columns should have proper dates.
5. Table names in the database should be unique. In the database, same schema cannot contain two or more tables with same name. But two tables with different names can have same column names. But same column name is not allowed in the same table.

Examine below table structure for Employee, Department and Project and see if it satisfies relational data model rules.



EMPLOYEE

| EMP_ID | EMP_NAME | ADDRESS | DEPT_ID | PROJ_ID |
|---|---|---|---|---|
| 100 | Joseph | Clinton Town | 10 | 206 |
| 101 | Rose | Fraser Town | 20 | 205 |
| 102 | Mathew | Lakeside Village | 10 | 206 |
| 103 | Stewart | Troy | 30 | 204 |
| 104 | William | Holland | 30 | 202 |

DEPARTMENT

| DEPT_ID | DEPT_NAME |
|---|---|
| 10 | Accounting |
| 20 | Quality |
| 30 | Design |

PROJECT

| PROJ_ID | PROJ_NAME |
|---|---|
| 201 | C Programming |
| 202 | Web development |
| 204 | Database Design |
| 205 | Testing |
| 206 | Pay Slip Generation |



Fig 1.10

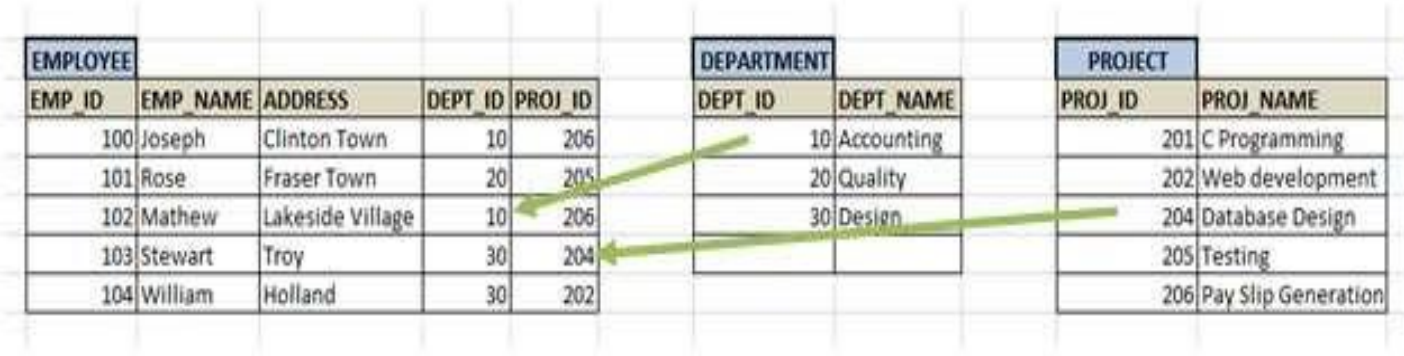
**Advantages**

- Structural independence: - Any changes to the database structure, does not the way we are accessing the data. For example, Age is added to Employee table. But it does not change the relationship between the other tables nor changes the existing data. Hence it provides the total independence from its structure.
- Simplicity: - This model is designed based on the logical data. It does not consider how data are stored physically in the memory. Hence when the designer designs the database, he concentrates on how he sees the data. This reduces the burden on the designer.
- Because of simplicity and data independence, this kind of data model is easy to maintain and access.
- This model supports structured query language – SQL. Hence it helps the user to retrieve and modify the data in the database. By the use of SQL, user can get any specific information from the database.

**Disadvantages**

Compared to the advantages above, the disadvantages of this model can be ignored.

- High hardware cost: - In order to separate the physical data information from the logical data, more powerful system hardware's – memory is required. This makes the cost of database high.
- Sometimes, design will be designed till the minute level, which will lead to complexity in the database.



EMPLOYEE

| EMP_ID | EMP_NAME | ADDRESS | DEPT_ID | PROJ_ID |
|---|---|---|---|---|
| 100 | Joseph | Clinton Town | 10 | 206 |
| 101 | Rose | Fraser Town | 20 | 205 |
| 102 | Mathew | Lakeside Village | 10 | 206 |
| 103 | Stewart | Troy | 30 | 204 |
| 104 | William | Holland | 30 | 202 |

DEPARTMENT

| DEPT_ID | DEPT_NAME |
|---|---|
| 10 | Accounting |
| 20 | Quality |
| 30 | Design |

PROJECT

| PROJ_ID | PROJ_NAME |
|---|---|
| 201 | C Programming |
| 202 | Web development |
| 204 | Database Design |
| 205 | Testing |
| 206 | Pay Slip Generation |

**Fig 1.11**

**Comparison of Record Based Model:**

| Hierarchical Data Model | Network Data Models | Relational Data Models |
|---|---|---|
| Supports One-Many Relationship | Supports both one to many and Many to Many relationship | Supports both one to many and Many to Many relationship |
| Because of single parent-child relationship, difficult to navigate through the child | It establishes the relationship between most of the objects, hence easy to access compared to hierarchical model | It provides SQL, which makes the access to the data simpler and quicker. |
| Flexibility among the different object is restricted to the child. | Because of the mapping among the sub level tables, flexibility is more | Primary and foreign key constraint makes the flexibility much simpler than other models. |
| Based on the physical storage details | Based on the physical storage details | Based on the logical data view |

**Relationship**

A relationship defines how two or more entities are inter-related. For example, STUDENT and CLASS entities are related as 'Student X **studies** in a Class Y'. Here 'Studies' defines the relationship between Student and Class. Similarly, Teacher and Subject are related as 'Teacher A **teaches** Subject B'. Here 'teaches' forms the relationship between both Teacher and Subject.

**Degrees of Relationship**

In a relationship two or more number of entities can participate. The number of entities who are part of a particular relationship is called degrees of relationship. If only two entities participate in the mapping, then degree of relation is 2 or binary. If three entities are involved, then degree of relation is 3 or ternary. If more than 3 entities are involved then the degree of relation is called n-degree or n-nary.

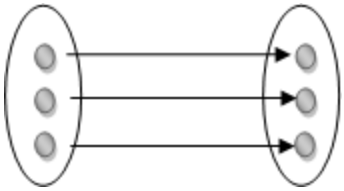
**Cardinality of Relationship**

How many number of instances of one entity is mapped to how many number of instances of another entity is known as cardinality of a relationship. In a 'studies' relationship above, what we observe is only one Student X is studying in on Class Y. i.e.; single instance of entity student mapped to a single instance of entity Class. This means the cardinality between Student and Class is 1:1.

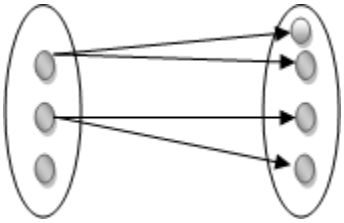
**Based on the cardinality, there are 3 types of relationship.**

- **One-to-One (1:1)**: As we saw in above example, one instance of the entity is mapped to only one instance of another entity.

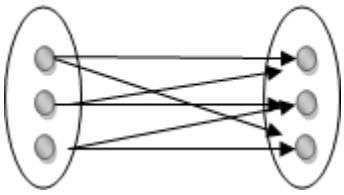
Consider, HOD of the Department. There is only one HOD in one department. That is there is 1:1 relationship between the entity HOD and Department.



• **One-to-Many (1: M)**: As we can guess now, one to many relationships has one instance of entity related to multiple instances of another entity. One manager manages multiple employees in his department. Here Manager and Employee are entities, and the relationship is one to many. Similarly, one teacher teaches multiple classes is also a 1: M relationship.



• **Many-to-Many (M: N)**: This is a relationship where multiple instances of entities are related to multiple instances of another entity. A relationship between TEACHER and STUDENT is many to many. How? Multiple Teachers teach multiple numbers of Students.
Similarly, above example of 1:1 can be M:N !! Surprised?? Yes, it can be M:N relationship, provided, how we relate these two entities. Multiple Students enroll for multiple classes/courses makes this relationship M:N. The relationship 'studies' and 'enroll' made the difference here. That means, it all depends on the requirement and how we are relating the entities.



**DBA Responsibilities**

- Maintaining all databases required for development, training, testing, and production usage
- Installation and configuration of DBMS server software and associated products.
- Upgrading and patching/hot-fixing of DBMS server software and related products.
- Migrate database to another server.
- Evaluate DBMS features and DBMS related products.
- Establish and maintain sound backup and recovery policies and procedures.
- Implement and maintain database security (create and maintain logins, users and roles, assign privileges).
- Performance tuning and health monitoring on DBMS, OS and application.
- Plan growth and changes (capacity planning).
- Do general technical troubleshooting and give consultation to development teams.
- Troubleshooting on DBMS and Operating System performance issue
- Documentation of any implementation and changes (database changes, reference data changes and application UI changes etc.)

**Types of DBA**

1. **Administrative DBA –** Work on maintaining the server and keeping it running. Concerned with installation, backups, security, patches, replication, OS configuration and tuning, storage management etc. Things that concern the actual server software.
2. **Development DBA** - works on building queries, stored procedures, etc. that meet business needs. This is the equivalent of the programmer. You primarily write T-SQL or PL-SQL.
3. **Database Architect** – Design schemas. Build tables, FKs, PKs, etc. Work to build a structure that meets the business needs in general. The design is then used by developers and development DBAs to implement the actual application.
4. **Data Warehouse DBA** - responsible for merging data from multiple sources into a data warehouse. May have to design warehouse, but cleans, standardizes, and scrubs data before loading. In SQL Server, this DBA would use SSIS heavily.
5. **OLAP DBA** – Builds multi-dimensional cubes for decision support or OLAP systems. The primary language in SQL Server is MDX, not SQL here
6. **Application DBA-** Application DBAs straddle the fence between the DBMS and the application software and are responsible for ensuring that the application is fully optimized for the database and vice versa. They usually manage all the application components that interact with the database and carry out activities such as application installation and patching, application upgrades, database cloning, building and running data cleanup routines, data load process management, etc.

**Generalization**

- It is a bottom-up approach in which two lower level entities combine to form higher entity. In generalization, the higher-level entity can also association with other lower level entity to make further higher-level entity.
- Generalization proceeds from the recognition that a number of entity sets share some common features. On the basis of the commonalities, generalization synthesizes these entity sets into a single, higher-level entity set.
- Generalization is used to emphasize the similarities among lower-level entity sets and to hide the differences in the schema.

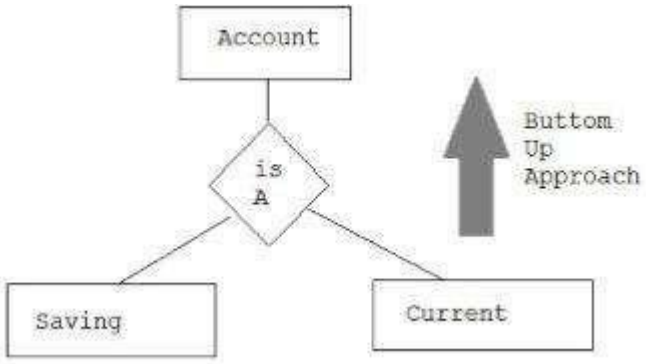



**Fig 1.13**

**Specialization**

- It is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into lower level entity.
- Example: The specialization of student allows us to distinguish among students according to whether they are Ex-Student or Current Student.
- Specialization can be repeatedly applied to refine a design schema.

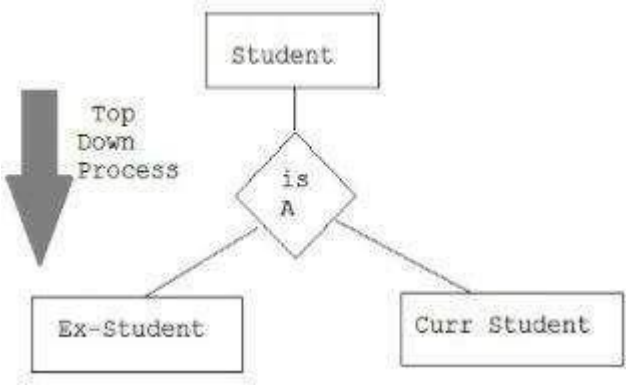



**Fig 1.14**

**Aggregation**

- One limitation of the E-R model is that it cannot express relationships among relationships. To illustrate the need for such a construct, quaternary relationships are used which lead to redundancy in data storage.
- The best way to mode such situations is to use aggregation.
- Aggregation is an abstraction through which relationships are treated as higher-level entities.
- Below is the example of aggregation relation between offer (which is binary relation between center and course) and visitor.

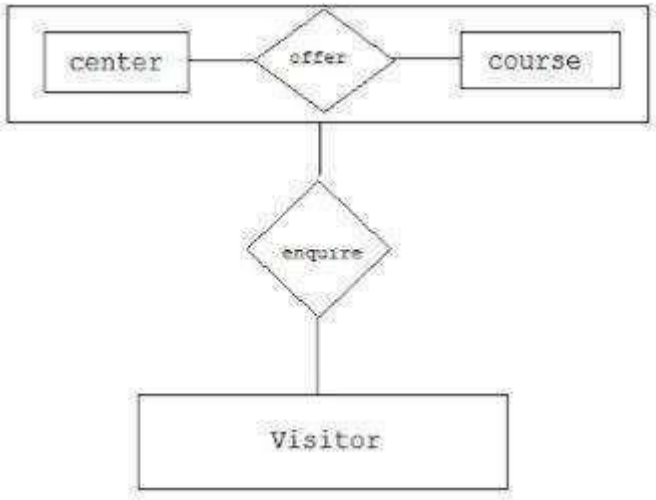



**Fig 1.15**

**Data Independence**

1. **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item). In the last case, external schemas that refer only to the remaining data should not be affected. Only the view definition and the mappings need to be changed in a DBMS that supports logical **data independence**. After the conceptual schema undergoes a logical reorganization, application programs that reference the external schema constructs must work as before. Changes to constraints can be applied to the conceptual schema without affecting the external schemas or application programs.

2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files were reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema. Generally, physical **data independence** exists in most databases and file environments where physical details such as the exact location

of data on disk, and hardware details of storage encoding, placement, compression, splitting, merging of records, and so on are hidden from the user. Applications remain unaware of these details. On the other hand, logical **data independence** is harder to achieve because it allows structural and constraint changes without affecting application programs a much stricter requirement.

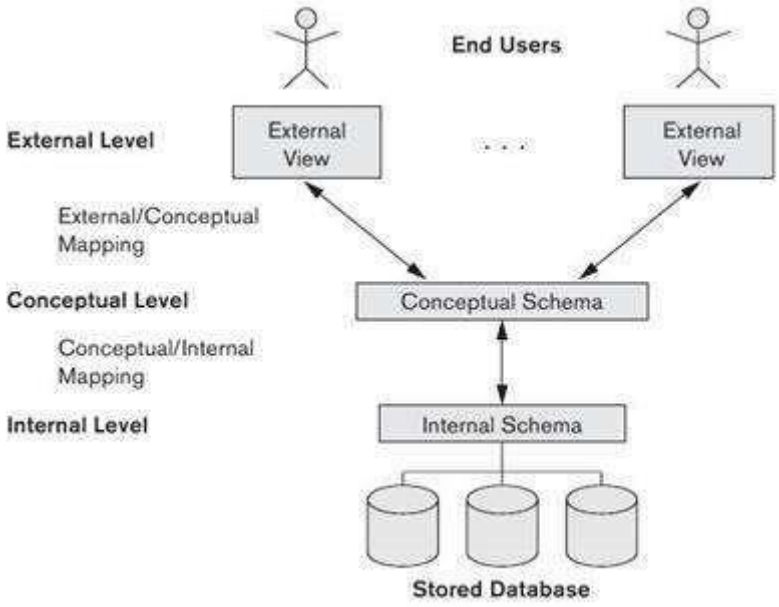



Fig 1.16

**Database Schema**

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



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

A database schema can be divided broadly into two categories −

**Physical Database Schema** − This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.

**Logical Database Schema** − This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

**Database Instance**

It is important that we distinguish these two terms individually. Database schema is the skeleton of database. It is designed when the database doesn't exist at all. Once the database is operational, it is very difficult to make any changes to it. A database schema does not contain any data or information.

A database instance is a state of operational database with data at any given time. It contains a snapshot of the database. Database instances tend to change with time. A DBMS ensures that its every instance (state) is in a valid state, by diligently following all the validations, constraints, and conditions that the database designers have imposed.

# Subject Notes (DBMS CS 502)

**UNIT-2**:

**Relational Data models:**

**Domain**: A (usually named) set/universe of *atomic* values, where by "atomic" we mean simply that, from the point of view of the database, each value in the domain is indivisible (i.e., cannot be broken down into component parts).

Examples of domains

- SSN: string of digits of length nine
- Name: string of characters beginning with an upper-case letter
- GPA: a real number between 0.0 and 4.0
- Sex: a member of the set {female, male}
- Dept_Code: a member of the set {CMPS, MATH, ENGL, PHYS, PSYC, …}

These are all *logical* descriptions of domains. For implementation purposes, it is necessary to provide descriptions of domains in terms of concrete **data types** (or **formats**) that are provided by the DBMS (such as String, int, Boolean), in a manner analogous to how programming languages have intrinsic data types.

**Attribute**: the *name* of the role played by some value (coming from some domain) in the context of a **relational schema**. The domain of attribute A is denoted Dom (A).

**Tuple**: A tuple is a mapping from attributes to values drawn from the respective domains of those attributes. A tuple is intended to describe some entity (or relationship between entities) in the inworld.

As an example, a tuple for a PERSON entity might be

{Name --> "Keerthy", Sex -->Male, IQ -->786}



**Relation**: A (named) set of tuples all of the same form (i.e., having the same set of attributes). The term **table** is a loose synonym.

- **Relational Schema**: used for describing (the structure of) a relation. E.g., R ($A_1$, $A_2$, ..., $A_n$) says that R is a relation with *attributes* $A_1$ ... $A_n$. The **degree** of a relation is the number of attributes it has, here *n*.

Example: STUDENT (Name, SSN, Address)

One would think that a "complete" relational schema would also specify the domain of each attribute.

- **Relational Database**: A collection of **relations**, each one consistent with its specified relational schema.

**Characteristics of Relations**

**Ordering of Tuples**: A relation is a *set* of tuples; hence, there is no order associated with them. That is, it makes no sense to refer to, for example, the 5th tuple in a relation. When a relation is depicted as a table, the tuples are necessarily listed in *some* order, of course, but you should attach no significance to that order. Similarly, when tuples are represented on a storage device, they must be organized in *some* fashion, and it may be advantageous, from a performance standpoint, to organize them in a way that depends upon their content.

**Ordering of Attributes**: A tuple is best viewed as a mapping from its attributes (i.e., the names we give to the roles played by the values comprising the tuple) to the corresponding values. Hence, the order in which the attributes are listed in a table is irrelevant. (Note that, unfortunately, the set theoretic operations in relational algebra (at least how Elmasri & Navathe define them) make implicit use of the order of the attributes. Hence, E & N view attributes as being arranged as a sequence rather than a set.)

The **Null** value: used for *don't know*, *not applicable*.

**Interpretation of a Relation**: Each relation can be viewed as a **predicate** and each tuple an assertion that that predicate is satisfied (i.e., has value **true**) for the combination of values in it. In other words, each tuple represents a fact. Keep in mind that some relations represent facts about entities whereas others represent facts about relationships (between entities).

**Relational database**

A relational database (RDB) is a collective set of multiple data sets organized by tables, records and columns. RDBs establish a well-defined relationship between database tables. Tables communicate and share information, which facilitates data searchability, organization and reporting.

RDBs use Structured Query Language (SQL), which is a standard user application that provides an easy programming interface for database interaction.

RDB is derived from the mathematical function concept of mapping data sets and was developed by Edgar F. Codd.
RDBs organize data in different ways. Each table is known as a relation, which contains one or more data category columns. Each table record (or row) contains a unique data instance defined for a corresponding column category. One or more data or record characteristics relate to one or many records to form functional dependencies. These are classified as follows:

- One to One: One table record relates to another record in another table.
- One to Many: One table record relates to many records in another table.
- Many to One: More than one table record relates to another table record.
- Many to Many: More than one table record relates to more than one record in another table.
- RDB performs "select", "project" and "join" database operations, where select is used for data retrieval, project identifies data attributes, and join combines relations.



RDBs advantages:

- Easy extendibility, as new data may be added without modifying existing records. This is also known as scalability.
- New technology performance, power and flexibility with multiple data requirement capabilities.
- Data security, which is critical when data sharing is based on privacy. For example, management may share certain data privileges and access and block employees from other data, such as confidential salary or benefit information.

**Database Schema**

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

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

A database schema can be divided broadly into two categories −

**Physical Database Schema** − This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.

**Logical Database Schema** − This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

**Keys** are the attributes of the entity, which uniquely identifies the record of the entity. For example, STUDENT_ID identifies individual students, passport#, license # etc.
As we have seen already, there are different types of keys in the database.
**Super Key** is the one or more attributes of the entity, which uniquely identifies the record in the database.
**Candidate Key** is one or more set of keys of the entity. For a person entity, his SSN, passport#, license# etc can be a super key.
**Primary Key** is the candidate key, which will be used to uniquely identify a record by the query. Though a person can be identified using his SSN, passport# or license#, one can choose any one of them as primary key to uniquely identify a person. Rest of them will act as a candidate key.

**Comparison of primary key and foreign key**

- Primary key is unique but foreign key need not be unique.
- Primary key is not null and foreign key can be null, foreign key references a primary key in another table.
- Primary key is used to identify a row; where as foreign key refers to a column or combination of columns.
- Primary key is the parent table and foreign key is a child table.

**Constraints**

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Relational Integrity Constraints**. There are three main integrity constraints −

- Key constraints
- Domain constraints
- Referential integrity constraints

**Key Constraints**

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key** for that relation. If there is more than one such minimal subset, these are called **candidate keys**.



Key constraints force that −

- In a relation with a key attribute, no two tuples can have identical values for key attributes.
- A key attribute cannot have NULL values.

Key constraints are also referred to as Entity Constraints.

**Domain Constraints**

Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.

**Referential integrity Constraints**

Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.
Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

**Intension and Extension-**

**Extension**

The set of tuples appearing at any instant in a relation is called the extension of that relation. In other words, instance of Schema is the extension of a relation. The extension varies with time as instance of schema or the value in the database will change with time.

**Intension**

The intension is the schema of the relation and thus is independent of the time as it does not change once created. So, it is the permanent part of the relation and consists of –

1. **Naming Structure –** Naming Structure includes the name of the relation and the attributes of the relation.
2. **Set of Integrity Constraints –** The Integrity Constraints are divided into Integrity Rule 1 (or entity integrity rule), Integrity Rule 2 (or referential integrity rule), key constraints, domain constraints etc.

For example:

**Employee (EmpNo Number (4) NOT NULL, EName Char(20), Age Number(2), Dept Char(4)**

**Relational Query languages-** Relational query languages use relational algebra to break the user requests and instruct the DBMS to execute the requests. It is the language by which user communicates with the database. These relational query languages can be procedural or non-procedural.

**Procedural query language** will have set of queries instructing the DBMS to perform various transactions in the sequence to meet the user request. For example, *get_CGPA* procedure will have various queries to get the marks of student in each subject, calculate the total marks, and then decide the CGPA based on his total marks. This procedural query language tells the database what is required from the database and how to get them from the database. Relational algebra is a procedural query language.

**Non-procedural queries** will have single query on one or more tables to get result from the database. For example, get the name and address of the student with particular ID will have single query on STUDENT table. Relational Calculus is a non-procedural language which informs what to do with the tables, but doesn't inform how to accomplish.

**SQL**

SQL language is divided into four types of primary language statements: DML, DDL, DCL and TCL. Using these statements, we can define the structure of a database by creating and altering database objects, and we can manipulate data in a table through updates or deletions. We also can control which user can read/write data or manage transactions to create a single unit of work.

The four main categories of SQL statements are as follows:



I. DDL (Data Definition Language)
II. DML (Data Manipulation Language)
III. DCL (Data Control Language)
IV. TCL (Transaction Control Language)

**DDL (Data Definition Language)**

DDL statements are used to alter/modify a database or table structure and schema. These statements handle the design and storage of database objects.

**CREATE** – create a new Table, database, schema

**Example:**

```
create table vendor_master(vencode varchar(5) unique, venname varchar(7) not null);
```

**ALTER** – alter existing table, column description

**Example:**

```
alter table vendor_masteradd(productprice int(10) check(productprice<10000));
```

**DROP** – delete existing objects from database

**Example:**

```
drop table vendor_master;
```

**DML (Data Manipulation Language)**

DML statements affect records in a table. These are basic operations we perform on data such as selecting a few records from a table, inserting new records, deleting unnecessary records, and updating/modifying existing records.
DML statements include the following:
**SELECT** – select records from a table

**Example:**
select * from order_master
**INSERT** – insert new records

**Example:**
insert into order_mastervalues('&oredrno','&odate','&vencode', '&o_status','&deldate');

**UPDATE** – update/Modify existing records

**Example:**
update person set address='United States'wherepid=5;

**DELETE** – delete existing records

**Example:**

delete from person where lastname='Kumar';



**DCL (Data Control Language)**
DCL statements control the level of access that users have on database objects.
**GRANT** – allows users to read/write on certain database objects

**Example:**
Grant select, update on dept to ABC;

**REVOKE** – keeps users from read/write permission on database objects

**Example:**
Reovek delete on emp form admin;

TCL (Transaction Control Language)
TCL statements allow you to control and manage transactions to maintain the integrity of data within SQL statements.
**BEGIN Transaction** – opens a transaction
**COMMIT Transaction** – commits a transaction

**Example:**
Set autocommit=0;

**ROLLBACK Transaction** – ROLLBACK a transaction in case of any error.

```
BEGIN TRAN @TransactionName
    INSERT INTO ValueTableVALUES(1), (2);
```

ROLLBACK TRAN @TransactionName;

**Complex queries-**

Complex SQL is the use of SQL queries which go beyond the standard SQL of using the SELECT and WHERE commands. Complex SQL often involves using complex joins and sub-queries, where queries are nested in WHERE clauses. Complex queries frequently involve heavy use of AND clauses and OR clauses. These queries make it possible for perform more accurate searches of a database.

**Various Joins: -**

Joins are operations that "cross reference" the data. That is, tuples from one relation are somehow matched with tuples from another relation to form a third relation. There are several types of joins, but the most basic type is the Cartesian join, sometimes called a Cartesian product or cross product. Other joins, including the natural join, the Equi-join and the theta join, are variations of the Cartesian join in which special rules are applied. Each of these four types of joins in described below.

**Indexing -**

Indexing is a way to optimize performance of a database by minimizing the number of disk accesses required when a query is processed. An index or database index is a data structure which is used to quickly locate and access the data in a database table.

Indexes are created using some database columns.

- The first column is the Search key that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly (Note that the data may or may not be stored in sorted order).
- The second column is the Data Reference which contains a set of pointers holding the address of the disk block where that particular key value can be found.

There are two kinds of indices:

1) **Ordered indices:** Indices are based on a sorted ordering of the values.
2) **Hash indices:** Indices are based on the values being distributed uniformly across a range of buckets. The bucket to which a value is assigned is determined by function called a hash function.

There is no comparison between both the techniques; it depends on the database application on which it is being applied.

- **Access Types**: e.g. value based search, range access, etc.
- **Access Time**: Time to find particular data element or set of elements.
- **Insertion Time**: Time taken to find the appropriate space and insert a new data time.
- **Deletion Time**: Time taken to find an item and delete it as well as update the index structure.
- **Space Overhead**: Additional space required by the index.

Indexing Methods

**Ordered Indices**

The indices are usually sorted so that the searching is faster. The indices which are sorted are known as ordered indices.

- If the search key of any index specifies same order as the sequential order of the file, it is known as primary index or clustering index.

**Note:** The search key of a primary index is usually the primary key, but it is not necessarily so.

- If the search key of any index specifies an order different from the sequential order of the file, it is called the secondary index or non-clustering index.

**Clustered Indexing**

Clustering index is defined on an ordered data file. The data file is ordered on a non-key field. In some cases, the index is created on non-primary key columns which may not be unique for each record. In such cases, in order to identify the records faster, we will group two or more columns together to get the unique values and

create index out of them. This method is known as clustering index. Basically, records with similar characteristics are grouped together and indexes are created for these groups.

**Primary Index**

In this case, the data is sorted according to the search key. It induces sequential file organization.

In this case, the primary key of the database table is used to create the index. As primary keys are unique and are stored in sorted manner, the performance of searching operation is quite efficient. The primary index is classified into two types: **Dense Index** and **Sparse Index**.

*(I) Dense Index:*

- For every search key value in the data file, there is an index record.
- This record contains the search key and also a reference to the first data record with that search key value.

*(II) Sparse Index:*

- The index record appears only for a few items in the data file. Each item points to a block as shown.
- To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.
- We start at that record pointed to by the index record, and proceed along the pointers in the file (that is, sequentially) until we find the desired record.

**Non-Clustered Indexing**

A non-clustered index just tells us where the data lies, i.e. it gives us a list of virtual pointers or references to the location where the data is actually stored. Data is not physically stored in the order of the index. Instead, data is present in leaf nodes. For example, the contents page of a book. Each entry gives us the page number or location of the information stored. The actual data here (information on each page of book) is not organized but we have an ordered reference (contents page) to where the data points actually lie.

**Secondary Index**

It is used to optimize query processing and access records in a database with some information other than the usual search key (primary key). In these two levels of indexing are used in order to reduce the mapping size of the first level and in general. Initially, for the first level, a large range of numbers is selected so that the mapping size is small. Further, each range is divided into further sub ranges.

In order for quick memory access, first level is stored in the primary memory. Actual physical location of the data is determined by the second mapping level.

**Trigger**

A **database trigger** is procedural code that is automatically executed in response to certain events on a particular table or view in a database. The trigger is mostly used for maintaining the integrity of the information on the database.

***Schema-level triggers.***

- After Creation
- Before Alter
- After Alter
- Before Drop
- After Drop
- Before Insert

The four main types of triggers are:

1. Row Level Trigger: This gets executed before or after *any column value of a row* changes
2. Column Level Trigger: This gets executed before or after the *specified column* changes

3. For Each Row Type: This trigger gets executed once for each row of the result set affected by an insert/update/delete
4. For Each Statement Type: This trigger gets executed only once for the entire result set, but also fires each time the statement is executed.

**Syntax**

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

**Details:**

CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the trigger_name.

{BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.

{INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.

[OF col_name] − This specifies the column name that will be updated.

[ON table_name] − This specifies the name of the table associated with the trigger.

[REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.

[FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.

WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

**Assertion**

An **assertion** is a statement in SQL that ensures a certain condition will always exist in the database. Assertions are like column and table constraints, except that they are specified separately from table definitions. An

example of a column constraint is NOT NULL, and an example of a table constraint is a compound foreign key, which, because it's compound, cannot be declared with column constraints.
Assertions are similar to check constraints, but unlike check constraints they are not defined on table or column level but are defined on schema level. (i.e., assertions are database objects of their own right and are not defined within a create table or alter table statement.)

**Relational Algebra**

Relational Algebra is a procedural language used for manipulating relations. The relational model gives the structure for relations so that data can be stored in that format but relational algebra enables us to retrieve information from relations. Some advanced SQL queries requires explicit relational algebra operations, most commonly outer join.
Relations are seen as sets of tuples, which means that no duplicates are allowed. SQL behaves differently in some cases. Remember the SQL keyword distinct. SQL is declarative, which means that you tell the DBMS what you want.

**Set operations**

Relations in relational algebra are seen as sets of tuples, so we can use basic set operations.
Review of concepts and operations from set theory

- Set Element
- No duplicate elements
- No order among the elements
- Subset
- Proper subset (with fewer elements)
- Superset
- Union
- Intersection
- Set Difference
- Cartesian product
- Relational Algebra
- Relational Algebra consists of several groups of operations



**Unary Relational Operations**

SELECT (symbol: s (sigma))
PROJECT (symbol: (pi))
RENAME (symbol: r (rho))

**Relational Algebra Operations from Set Theory**

UNION (U ), INTERSECTION ( ), DIFFERENCE (or MINUS, – )
CARTESIAN PRODUCT ( x )

**Binary Relational Operations**

JOIN (several variations of JOIN exist)

**Additional Relational Operations**

OUTER JOINS, OUTER UNION
AGGREGATE FUNCTIONS

**Unary Relational Operations**

SELECT (symbol: s (sigma))

**Selection**

Selection is another unary operation. It is performed on a single relation and returns a set of tuples with the same schema as the original relation. The selection operation must include a condition, as follows:

A selection of Relation A is a new relation with all of the tuples from Relation A that meet a specified condition. The condition must be a logical comparison of one or more of the attributes of Relation A and their possible values. Each tuple in the original relation must be checked one at a time to see if it meets the condition. If it does, it is included in the result set, if not, it is not included in the result set. The logical comparisons are the same as those used in Boolean conditions in most computer programming languages.
The symbol for selection is σ, a lowercase Greek letter sigma.
A selection operation is written as $B = \sigma_c(A)$ , where c is the condition.

**Example:**
We wish to find all members of the U.S. Supreme Court born before 1935 in Arizona

Let A = the relation containing data on members of the Supreme Court, as defined above.

$B = \sigma_{((\text{year of birth} < 1935) \wedge (\text{state of birth} = \text{“Arizona”}))}(A)$.

B = {(William, Rehnquist, Arizona, 1924), (Sandra, O'Connor, Arizona, 1930)}

The allowable operators for the logical conditions are the six standard logical comparison operators: equality, inequality, less than, greater than, not less than (equality or greater than), not greater than (equality or less than) Simple logical comparisons may be connected via conjunction, disjunction and negation (*and*, *or*, and *not*).

The symbols for the six logical comparison operators are:

| **Comparison Operation** | **Symbol** |
|---|---|
| equal to | = |
| not equal to | ≠ or<> |
| less than | < |
| greater than | > |
| not less than (greater than or equal to) | ≥ or >= |
| not greater than (less than or equal to) | ≤ or <= |

The symbols for the three logical modifiers used to build complex conditions are:

| **Modifier** | **Symbol** |
|---|---|
| And | ∧ |
| or | ∨ |
| not | ¬ or ~ |

Some definition or collating sequence must exist to determine how values compare to one another for each of the data types used, just as in computer programming languages. For example, 2 comes before 11 if the values are integers, but “11” comes before “2” if the values are character strings.

Each time we perform a selection operation, the result set will have the same number or fewer tuples. Most often, the result set gets smaller with each selection operation.
σsubject = "database"(Books)
Output − Selects tuples from books where subject is 'database'.

σsubject = "database" and price = "450"(Books)
Output − Selects tuples from books where subject is 'database' and 'price' is 450.

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

**Projection**

Projection is another unary operation, performed on a single relation with a result set that is a single relation, defined as follows:
The projection operation returns a result set with all rows from the original relation, but only those attributes that are specified in the projection.

Projection is shown by $\prod$ , the upper case Greek letter Pi. A selection operation is written as $B = \prod_{attributes} (A)$, where $_{attributes}$ is a list of the attributes to be included in the result set.

Example:
We wish to show only the names of the U. S. Supreme Court Justices from the example above.

Let A = the relation containing data on members of the Supreme Court, as defined above.

$B = \prod_{first\ name,\ last\ name} (A)$



B = { (William, Rehnquist), (John, Stevens), (Sandra, O'Connor), (Antonin, Scalia), (Anthony, Kennedy), (David, Souter), (Clarence, Thomas), (Ruth, Ginsburg), (Stephen, Breyer) }

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

A projection operation will return the same number of tuples, but with a new schema that will include either the same columns or fewer columns. Most often, the number of columns shrinks with each projection.

**Combining Selection and Projection**
Often the selection and projection operations are combined to select certain data from a single relation. We can nest the operation with parenthesis, just like in ordinary algebra.

Example:
We wish to show only the names of the U.S. Supreme Court justices born in Arizona before 1935.

Let A = the relation containing data on members of the U. S. Supreme Court, as defined above.

$B = \prod_{first\ name,\ last\ name} (\sigma_{((year\ of\ birth < 1935) \wedge (state\ of\ birth = \text{“Arizona”}))} (A))$

B = { (William, Rehnquist), (Sandra, O'Connor) }

When performing both a projection and a selection, which would be more efficient to do first, a projection or a selection? Imagine that we have a database of 100,000 student records. We wish to find the student #, name, and GPA for student # 111-11-1111. Should we find the student's record first and then pull out the name and GPA, or should we pull out the name and GPA for all students, then search that set for the student we are seeking? Usually it is best to do the selection first, thereby limiting the number of tuples, but this is not always the case. Fortunately, most good database management systems have optimizing compilers that will perform the operations in the most efficient way possible.

The most common queries on single tables in modern data base management systems are equivalent to a combination of the selection and projection operations.

**Union**

In simple set theory, the union of Set A and Set B includes all of the elements of Set A and all of the elements of Set B. In relational algebra, the union operation is similar:

The Union of relation A and relation B is a new relation containing all of the tuples contained in either relation A or relation B. Union can only be performed on two relations that have the same schema.

The symbol for union is ∪In relational algebra we would write something like R3 = R1 ∪R2.

**Example:**

The set of students majoring in Communications includes all of the Acting majors and all of the Journalism majors.

Let A = the relation with data for all Acting majors

Let J = the relation with data for all Journalism majors

Let C = the relation with data for all Communications majors



C = A ∪J

The union operation is both commutative and associative.

Commutative law of union: A ∪B = B ∪A

Associative law of union: (A ∪B) ∪C = A ∪(B ∪C)

**Intersection**

Like union, intersection means pretty much the same thing in relational algebra that it does in simple set theory:

The Intersection of relation A and relation B is a new relation containing all of the tuples that are contained in both relation A and relation B. Intersection can only be performed on two relations that have the same schema.

The symbol for intersection is ∩ **In** relational algebra we would write something like R3 = R1 ∩R2.

Example:

We might want to define the set of all students registered for both Database Management and Linear Algebra.

Let D = the relation with data for all students registered for Database Management

Let L = the relation with data for all students registered for Linear Algebra

Let B = the relation with data for all students registered for both Database Management and Linear Algebra

B = D ∩L

The intersection operation is both commutative and associative.

Commutative law of intersection: $A \cap B = B \cap A$
Associative law of intersection: $(A \cap B) \cap C = A \cap (B \cap C)$
Unlike union, however, intersection is not considered a basic operation, but a derived operation, because it can be derived from the basic operations. We will look at difference next, but the derivation looks like this:

$R1 \cap R2 = R1 - (R1 - R2)$

For our purposes, however, it really doesn't matter that intersection is a derived operation.

**Difference**
The difference operation also means pretty much the same thing in relational algebra that it does in simple set theory:
The difference between relation A and relation B is a new relation containing all of the tuples that are contained in relation A but not in relation B. Difference can only be performed on two relations that have the same schema.
The symbol for difference is the same as the minus sign - We would write R3 = R1 – R2.

**Example:**
We might want to define the set of all players sitting on the bench during a basketball game.

Let T = the relation with data for all players currently on the team
Let G = the relation with data for all players currently in the game
Let B = the relation with data for all players on the team but not playing



B = T - G

The difference operation is neither commutative nor associative.

$A - B \neq B - A$
$(A - B) - C \neq A - (B - C)$

**Complement**
Union, Intersection, and difference were *binary* operations; that is, they were performed on two relations with the result being a third relation. Complement is a *unary* operation; it is performed on a single relation to form a new relation, as follows:
The complement of relation A is a relation composed all possible tuples not in A, which have the same schema as A, derived from the same range of values for each attribute of A.
Complement is sometimes shown by using a superscripted *C*, like this: $B = A^C$.

Example:
Let A = the relation containing data on members of the U.S. Supreme Court, with the schema (first name, last name, state of birth, year of birth).

A = { (William, Rehnquist, Arizona, 1924), (John, Stevens, Illinois, 1920), (Sandra, O'Connor, Arizona, 1930), (Antonin, Scalia, New Jersey, 1936), (Anthony, Kennedy, California, 1936), (David, Souter, Massachusetts, 1939), (Clarence, Thomas, Georgia, 1948), (Ruth, Ginsburg, New York, 1933), (Stephen, Breyer, California,

1938) }

$A^C$ would be the set of all possible tuples with the same schema as A but not in A, that are derived from the same domains for the attributes. It would be very large and include tuples like (William, Rehnquist, Arizona, 1920), (William, Rehnquist, Arizona, 1930), (Ruth, Rehnquist, Illinois, 1924), (William, Stevens, Illinois, 1930), (John, O'Connor, Georgia, 1938), (Clarence, Ginsberg, California, 1936), and so on.

If we perform the complement operation twice, we get back the original relation, just like we would when using the negation operation on numbers in simple arithmetic. $(A^C)^C = A$, which means that if $B = A^C$, then $A = B^C$.

**Joins**

Joins are operations that "cross reference" the data. That is, tuples from one relation are somehow matched with tuples from another relation to form a third relation. There are several types of joins, but the most basic type is the Cartesian join, sometimes called a Cartesian product or cross product. Other joins, including the natural join, the equi-join and the theta join, are variations of the Cartesian join in which special rules are applied. Each of these four types of joins in described below.

**Cartesian Join**

Imagine that we have two sets, one composed of letters and one composed of numbers, as follows:

S1 = { a, b, c, d} and S2 = { 1,2,3}

The cross product of the two sets is a set of ordered pairs, matching each value from S1 with each value from S2.



S1 x S2 = { (a,1), (a2), (a3), (b1), (b2), (b3), (c,1), (c2), (c3), (d1), (d2), (d3) }

The cross product of two sets is also called the Cartesian product, after René Descartes, the French mathematician and philosopher, who among other things, developed the Cartesian Coordinates used in quantifying geometry. In Cartesian coordinates, we have an X-axis and a Y-axis, which means we have a set of X values and a set of Y values. Each point on the Cartesian plane can be referenced by its coordinates, with an X-Y ordered pair: (x,y). The set of all possible X and Y coordinates is the cross product of the set of all X coordinates with the set of all Y coordinates.

In relational algebra, the cross product of two relations is also called the Cartesian Product or Cartesian Join, and is defined as follows:

The Cartesian Join of relation A and relation B is composed by matching each tuple from relation A one at time, with each tuple from relation B one at a time to form a new relation containing tuples with all of the attributes from tuple A and all of the attributes from tuple B. If tuple A has $A_T$ tuples and AA attributes and tuple B has $B_T$ tuples and $B_A$ attributes, then the new relation will have $(A_T * B_T)$ tuples, and $(A_A + B_A)$ attributes.

The symbol for a Cartesian Join is $\times$ We would write $C = A \times B$.

**Example:**

We wish to match a group of drivers with a group of trucks.

Let D = the relation with data for all of the drivers
D has the schema D(name, years of service)

D = { (Joe Smith, 12), (Mary Jones, 4), (Sam Wilson, 20), Bob Johnson, 8) }

Let T = the relation with data for all of the trucks
T has the schema D (make, model, year purchased)
T = {(White, Freightliner, 1996), (Ford, Econoline, 2002), (Mack, CHN 602, 2004)}

Let A = the relation with data for all possible assignments of driver to trucks
$A = D \times T$
A has the schema A (name, years of service, make, model, year purchased)

A = { (Joe Smith, 12, White, Freightliner, 1996), (Joe Smith, 12, Ford, Econoline, 2002), (Joe Smith, 12, Mack, CHN 602, 2004 ), (Mary Jones, 4, White, Freightliner, 1996), (Mary Jones, 4, Ford, Econoline, 2002), (Mary Jones, 4, Mack, CHN 602, 2004 ), (Sam Wilson, 20, White, Freightliner, 1996), (Sam Wilson, 20, Ford, Econoline, 2002), (Sam Wilson, 20, Mack, CHN 602, 2004 ), (Bob Johnson, 8, White, Freightliner, 1996), (Bob Johnson, 8, Ford, Econoline, 2002), (Bob Johnson, 8, Mack, CHN 602, 2004 ) }

Although Cartesian Joins form the conceptual basis for all other joins, they are rarely used in actual database management systems because they often result in a relation with a large amount of data. Consider the case of a table with data for 40,000 students, with each row needing 300 bytes of storage space, and a table for 2,000 advisors, with each row needing 200 bytes. The two original tables would need about 12,000,000 and 400,000 bytes of storage space (12 megabytes and 400 kilobytes). The Cartesian join of these two would have 80,000,000 records, each with nearly 600 bytes of storage space for a total of 48,000,000,000 bytes (48 gigabytes).
Another reason that Cartesian joins are not used often is this: What is the value of a Cartesian join? How often do we really need to create such a table?

The other types of joins, which are based on the Cartesian join, are used more often, and are commonly applied in combination with projection and selection operations.

**Natural Join**
A natural join is performed on two relations that share at least one attribute, and is defined as follows:
The natural join of relation A with relation B is a new relation formed by matching all tuples from relation A one by one with all tuples from relation B one by one, but only where the value of the shared attributes are the same. Each shared attribute is only included once in the schema of the result set. A natural join can only be performed on two relations that have at least one shared attribute.
The symbol for a natural join is $\bowtie$We would write $C = A \bowtie B$.

**Theta Join**
A theta join is similar to a Cartesian join, except that only those tuples are included that meet a specified condition, as follows:
The theta join of relation A with relation B is a new relation formed by matching all tuples from relation A one by one with all tuples from relation B one by one, but only where the tuples meet a specified condition, called the theta predicate. If the relations share any attributes, then each shared attribute is only included once in the schema of the result set.
The general symbol for a theta join is composite symbol, similar to the symbol for a natural join subscripted with the Greek letter theta: $\bowtie_\theta$ We would write $C = A \bowtie_\theta B$. In practice, the theta is replaced with the actual condition.

**Equi-Join**
An equi-join, which is similar to both a theta joins and a natural join, is defined as follows:

The equi-join of relation A with relation B is a new relation formed by matching all tuples from relation A one by one with all tuples from relation B one by one, but only where the tuples meet a specified condition of equality, called the equi-join predicate. If the relations share any attributes, then each shared attribute is only included once in the schema of the result set.

The symbolism for an equi-join is similar to the symbol for a natural join subscripted with the equal sign: $\bowtie_{=}$ We would write $C = A \bowtie_{=} B$. Just as with the theta join, in practice the equal sign is replaced with the actual condition.

The difference between an equi-join and a theta join is that the condition must be one of equality in an equi-join. The difference between an equi-join and a natural join is that the two relations do not need to have a common attribute in an equi-join.

**Example:**
We wish to match groups of people waiting for tables at a restaurant with the available tables, on the condition that the number of people in the group equals the number of seats at the table.

Let W = the relation with data for all the groups waiting for tables
Let T = the relation with data for all of the available tables
Let M = the relation with data assigning groups to tables

$M = W \bowtie_{group.size = table.seats} T$

In this notation the attribute names are shown in their more complex form, *relation.attribute*, so that *group.size* refers to the *size* attribute of the *group* relation, and *table.seats* refers to the *seats* attribute of the *table* relation.



**Summary of Relation Algebra:**

| OPERATION | 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. | π<attribute list>(R) |
| THETA JOIN | Produces all combinations of tuples from R and R1 2 that satisfy the join condition. | R1 <join condition> R2 |
| EQUIJOIN | Produces all the combinations of tuples from R1 and R2 that satisfy a join condition with only equality comparisons. | R1 <join condition> R2, OR (<join attributes 1>), (<join attributes 2>) R2 |

| | | |
|---|---|---|
| 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 | R1*<join condition> R2, OR R1* (<join attributes 1>), (<join attributes 2>) R2 OR R1 * R2 |
| 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 ∪ R2 |
| INTERSECTION | Produces a relation that includes all the tuples in both R1 and R2; R1 and R2 must be union-compatible. | R1 ∩ R2 |
| 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 ∪ Y. | R1(Z) ÷ R2(Y) |



**Different Types of constraints in DBMS with Example:**

- Domain Constraints
- Tuple Uniqueness Constraints
- Key Constraints
- Single Value Constraints
- Integrity Rule 1 (Entity Integrity Rule or Constraint)
- Integrity Rule 2 (Referential Integrity Rule or Constraint)
- General Constraints

***Domain Constraints –***

Domain Constraints specifies that what set of values an attribute can take. Value of each attribute X must be an atomic value from the domain of X.

The data type associated with domains include integer, character, string, date, time, currency etc. An attribute value must be available in the corresponding domain. Consider the example below –

| SID | Name | Class(Sem) | Age |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 8001 | Ankit | 1st | 19 |
| 8002 | Srishti | 1st | 18 |
| 8003 | Somvir | 4th | 22 |
| 8004 | Sourabh | 6th | A |

**Not Allowed. Because age in an Integer attribute.**

***Tuple Uniqueness Constraints –***

A relation is defined as a set of tuples. All tuples or all rows in a relation must be unique or distinct. Suppose if in a relation, tuple uniqueness constraint is applied, then all the rows of that table must be unique i.e. it does not contain the duplicate values. For example,

| SID | Name | Class (semester) | Age |
|---|---|---|---|
| 8001 | Ankit | 1st | 19 |
| 8002 | Srishti | 2nd | 18 |
| 8003 | Somvir | 4th | 22 |
| 8004 | Sourabh | 6th | 19 |

Not Allowed, beacues all rows must be unique

***Key Constraints –***

Keys are attributes or sets of attributes that uniquely identify an entity within its entity set. An Entity set E can have multiple keys out of which one key will be designated as the primary key. Primary Key must have unique and not null values in the relational table. In an subclass hierarchy, only the root entity set has a key or primary key and that primary key must serve as the key for all entities in the hierarchy.

Example of Key Constraints in a simple relational table –

| SID | Name | Class (semester) | Age |
|---|---|---|---|
| 8001 | Ankit | 1st | 19 |
| 8002 | Srishti | 1st | 18 |
| 8003 | Somvir | 4th | 22 |
| 8004 | Sourabh | 6th | 45 |
| 8002 | Tony | 5th | 23 |

Not allowed as Primary key values must be unique

***Single Value Constraints –***

Single value constraints refer that each attribute of an entity set has a single value. If the value of an attribute is missing in a tuple, then we can fill it with a "null" value. The null value for a attribute will specify that either the value is not known or the value is not applicable. Consider the below example-

| SID | Name | Class (semester) | Age | Driving License Number |
|---|---|---|---|---|
| 8001 | Ankit | 1st | 19 | DL-45698 |
| 8002 | Srishti | 2nd | 18 | DL-45871, DL-89740 |
| 8003 | Somvir | 4th | 22 | DL-95687 |
| 8004 | Sourabh | 6th | 19 | |

Not allowed as a person does not have two driving licenses.

allowed as person may or may not have driving license

***Integrity Rule 1 (Entity Integrity Rule or Constraint)***

The Integrity Rule 1 is also called Entity Integrity Rule or Constraint. This rule states that no attribute of primary key will contain a null value. If a relation has a null value in the primary key attribute, then uniqueness property of the primary key cannot be maintained. Consider the example below-

| SID | Name | Class (semester) | Age |
|---|---|---|---|
| 8001 | Ankit | 1st | 19 |
| 8002 | Srishti | 2nd | 18 |
| 8003 | Somvir | 4th | 22 |
| | Sourabh | 6th | 19 |

Not allowed as primary key cannot contain a NULL value

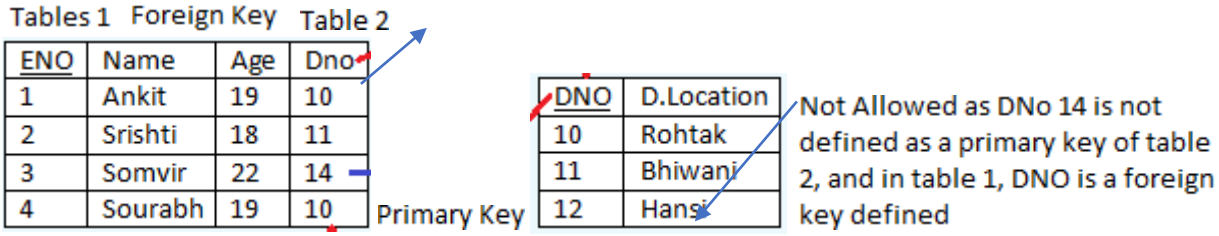
***Integrity Rule 2 (Referential Integrity Rule or Constraint) –***

The integrity Rule 2 is also called the Referential Integrity Constraints. This rule states that if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2. For example,

Some more Features of Foreign Key –

Let the table in which the foreign key is defined is Foreign Table or details table i.e. Table 1 in above example and the table that defines the primary key and is referenced by the foreign key is master table or primary table i.e. Table 2 in above example. Then the following properties must be hold:

- Records cannot be **inserted** into a F**oreign table** if corresponding records in the master table do not exist.
- Records of the **master table or Primary Table** cannot be **deleted** or **updated** if corresponding records in the detail table actually exist.




**Relational calculus**

**Relational calculus** is a non-procedural query language. It uses mathematical predicate calculus instead of algebra. It provides the description about the query to get the result whereas relational algebra gives the method to get the result. It informs the system what to do with the relation, but does not inform how to perform it.

For example, steps involved in listing all the students who attend 'Database' Course in relational algebra would be

- SELECT the tuples from COURSE relation with COURSE_NAME = 'DATABASE'
- PROJECT the COURSE_ID from above result
- SELECT the tuples from STUDENT relation with COUSE_ID resulted above.

In the case of relational calculus, it is described as below:

Get all the details of the students such that each student has course as 'Database'.

See the difference between relational algebra and relational calculus here. From the first one, we are clear on how to query and which relations to be queried. But the second tells what needs to be done to get the students with 'database' course. But it does tell us how we need to proceed to achieve this. Relational calculus is just the explanative way of telling the query.

There are two types of relational calculus - Tuple Relational Calculus (TRC) and Domain Relational Calculus (DRC).

**Tuple Relational Calculus**

A tuple relational calculus is a non-procedural query language which specifies to select the tuples in a relation. It can select the tuples with range of values or tuples for certain attribute values etc. The resulting relation can have one or more tuples. It is denoted as below:

**{t | P (t)}** or **{t | condition (t)}** -- this is also known as expression of relational calculus

Where t is the resulting tuples, P(t) is the condition used to fetch t.

**{t | EMPLOYEE (t) and t.SALARY>10000}** - implies that it selects the tuples from EMPLOYEE relation such that resulting employee tuples will have salary greater than 10000. It is example of selecting a range of values.

**{t | EMPLOYEE (t) AND t.DEPT_ID = 10}** – this select all the tuples of employee name who work for Department 10.

The variable which is used in the condition is called **tuple variable**. In above example t.SALARY and t.DEPT_ID are tuple variables. In the first example above, we have specified the condition t.SALARY>10000. What is the meaning of it? For all the SALARY>10000, display the employees. Here the SALARY is called as bound variable. Any tuple variable with '*For All*' (?) or '*there exists*' (ʔ) condition is **called bound variable**. Here, for any range of values of SALARY greater than 10000, the meaning of the condition remains the same. Bound variables are those ranges of tuple variables whose meaning will not change if the tuple variable is replaced by another tuple variable.

In the second example, we have used DEPT_ID= 10. That means only for DEPT_ID = 10 display employee details. Such variable is called free variable. Any tuple variable without any '*For All*' or '*there exists*' condition is called **Free Variable**. If we change DEPT_ID in this condition to some other variable, say EMP_ID, the meaning of the query changes. For example, if we change EMP_ID = 10, then above it will result in different result set. Free variables are those ranges of tuple variables whose meaning will change if the tuple variable is replaced by another tuple variable.

All the conditions used in the tuple expression are called as well formed formula – WFF. All the conditions in the expression are combined by using logical operators like AND, OR and NOT, and qualifiers like 'For All' (?) or 'there exists' (?). If the tuple variables are all bound variables in a WFF is called **closed WFF**. In an **open WFF**, we will have at least one free variable.

**Domain Relational Calculus**

In contrast to tuple relational calculus, domain relational calculus uses list of attributes to be selected from the relation based on the condition. It is same as TRC, but differs by selecting the attributes rather than selecting whole tuples. It is denoted as below:

$\{< a_1, a_2, a_3, \ldots a_n > \mid P(a_1, a_2, a_3, \ldots a_n)\}$

Where $a_1, a_2, a_3, \ldots a_n$ are attributes of the relation and P is the condition.

For example, select EMP_ID and EMP_NAME of employees who work for department 10

**{<EMP_ID, EMP_NAME> | <EMP_ID, EMP_NAME> ? EMPLOYEE Λ DEPT_ID = 10}**

Get name of the department name that Alex works for.

**{DEPT_NAME |< DEPT_NAME > ? DEPT Λ ? DEPT_ID (<DEPT_ID> ? EMPLOYEE Λ EMP_NAME = Alex)}**

Here green color expression is evaluated to get the department Id of Alex and then it is used to get the department name form DEPT relation.

Let us consider another example where select EMP_ID, EMP_NAME and ADDRESS the employees from the department where Alex works. What will be done here?

**{<EMP_ID, EMP_NAME, ADDRESS, DEPT_ID > | <EMP_ID, EMP_NAME, ADDRESS, DEPT_ID> ? EMPLOYEE Λ ? DEPT_ID (<DEPT_ID> ? EMPLOYEE Λ EMP_NAME = Alex)}**

First, formula is evaluated to get the department ID of Alex (green color), and then all the employees with that department is searched (red color).

Other concepts of TRC like free variable, bound variable, WFF etc. remains same in DRC too. Its only difference is DRC is based on attributes of relation.