DATA BASE MANAGEMENT SYSTEM

(M.Sc.(CA)-103

SELF LEARNING MATERIAL

DIRECTORATE

OF DISTANCE EDUCATION

SWAMI VIVEKANAND SUBHARTI UNIVERSITY

MEERUT – 250 005,

UTTAR PRADESH (INDIA)

SLM Module Developed By :

Author:

Reviewed by :

Assessed by:

Study Material Assessment Committee, as per the SVSU ordinance No. VI (2)

Published by: Gayatri Sales

Typeset at: Micron Computers **Printed at:** Gayatri Sales, Meerut.

DATA BASE MANAGEMENT SYSTEM

(M.Sc.(CA)-103

Unit - I

Overview of Database Management System

Elements of Database System, DBMS and its architecture, Advantage of DBMS (including Data independence), Types of database users, Role of Database administrator.

Unit - II

Data Models

Brief overview of Hierarchical and Network Model, Detailed study of Relational Model (Relations, Properties, Key & Integrity rules), Comparison of Hierarchical, Network and Relational Model ,CODD's rules for Relational Model,E-R diagram.

Unit - III

Normalization

Normalization concepts and update anomalies ,Functional dependencies,Multivalued and join dependencies.

Normal Forms: (1 NF, 2 NF, 3NF, BCNF, 4NF, and 5NF)

Unit - IV

SQL

SQL Constructs, SQL Join: Multiple Table Queries, Build-in functions, Views and their use, Overviews of ORACLE: (Data definition and manipulation)

Unit - V

Database Security, Integrity and Control

Security and Integrity threats, Defense mechanism, Integrity, Auditing and Control, Recent trends in DBMS- Distributed and Deductive Database.

UNIT – I

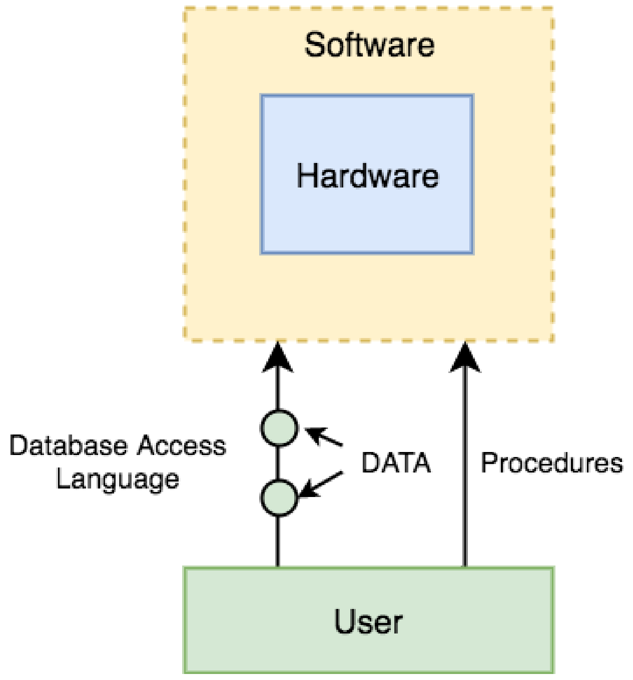
Overview of Database Management System

Elements of Database System:-

The database management system can be divided into five major components, they are:

1. Hardware
2. Software
3. Data
4. Procedures
5. Database Access Language

Let's have a simple diagram to see how they all fit together to form a database management system.



DBMS Components: Hardware

When we say Hardware, we mean computer, hard disks, I/O channels for data, and any other physical component involved before any data is successfully stored into the memory.

When we run Oracle or MySQL on our personal computer, then our computer's Hard Disk, our Keyboard using which we type in all the commands, our computer's RAM, ROM all become a part of the DBMS hardware.

DBMS Components: Software

This is the main component, as this is the program which controls everything. The DBMS software is more like a wrapper around the physical database, which provides us with an easy-to-use interface to store, access and update data.

The DBMS software is capable of understanding the Database Access Language and intrepret it into actual database commands to execute them on the DB.

DBMS Components: Data

Data is that resource, for which DBMS was designed. The motive behind the creation of DBMS was to store and utilise data.

In a typical Database, the user saved Data is present and **meta data** is stored.

Metadata is data about the data. This is information stored by the DBMS to better understand the data stored in it.

For example: When I store my **Name** in a database, the DBMS will store when the name was stored in the database, what is the size of the name, is it stored as related data to some other data, or is it independent, all this information is metadata.

DBMS Components: Procedures

Procedures refer to general instructions to use a database management system. This includes procedures to setup and install a DBMS, To login and logout of DBMS software, to manage databases, to take backups, generating reports etc.

DBMS Components: Database Access Language

Database Access Language is a simple language designed to write commands to access, insert, update and delete data stored in any database.

A user can write commands in the Database Access Language and submit it to the DBMS for execution, which is then translated and executed by the DBMS.

User can create new databases, tables, insert data, fetch stored data, update data and delete the data using the access language.

Users

- **Database Administrators:** Database Administrator or DBA is the one who manages the complete database management system. DBA takes care of the security of the DBMS, it's availability, managing the license keys, managing user accounts and access etc.
- **Application Programmer or Software Developer:** This user group is involved in developing and desiging the parts of DBMS.
- **End User:** These days all the modern applications, web or mobile, store user data. How do you think they do it? Yes, applications are programmed in such a way that they collect user data and store the data on DBMS systems running on their server. End users are the one who store, retrieve, update and delete data.

DBMS and its architecture

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

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

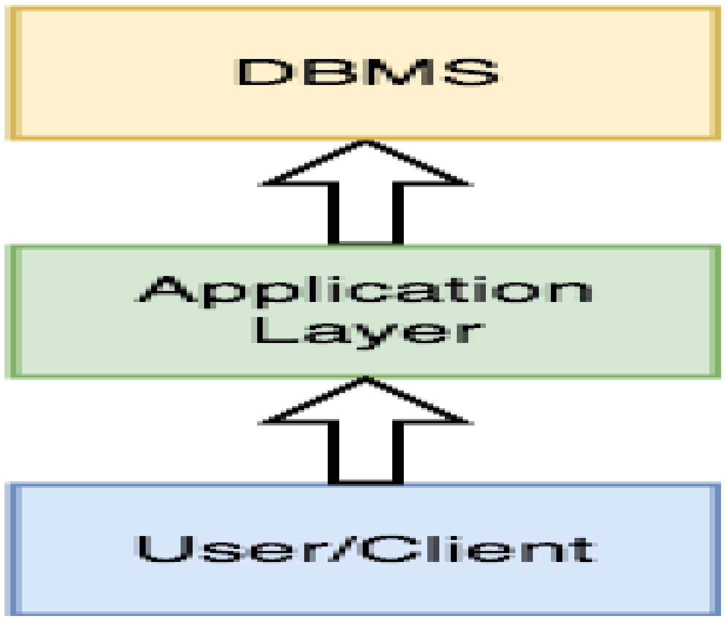
Database Architecture is logically of two types:

1. 2-tier DBMS architecture
2. 3-tier DBMS architecture

2-tier DBMS Architecture

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

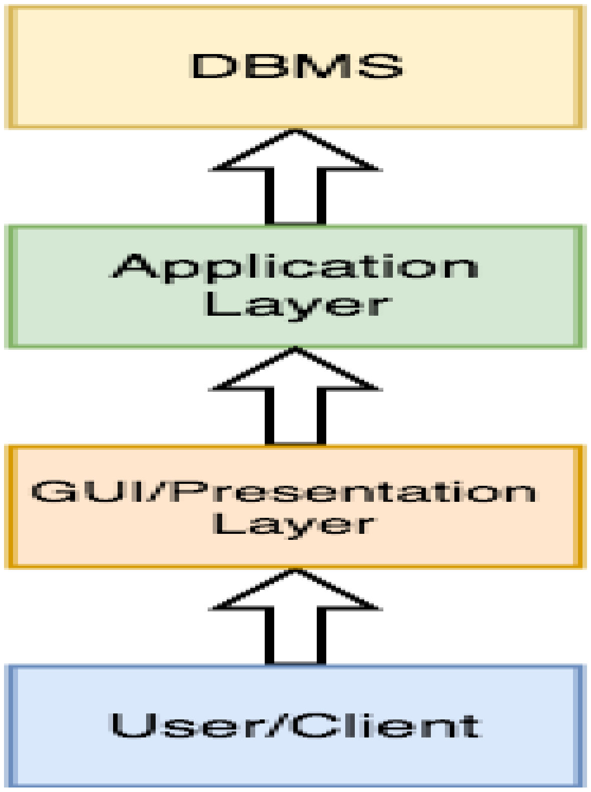
An application interface known as **ODBC**(Open Database Connectivity) provides an API that allow client side program to call the DBMS. Most DBMS vendors provide ODBC drivers for their DBMS.



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

3-tier DBMS Architecture

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



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

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

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

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

DBMS Database Models

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

- Hierarchical Model
- Network Model
- Entity-relationship Model

- Relational Model

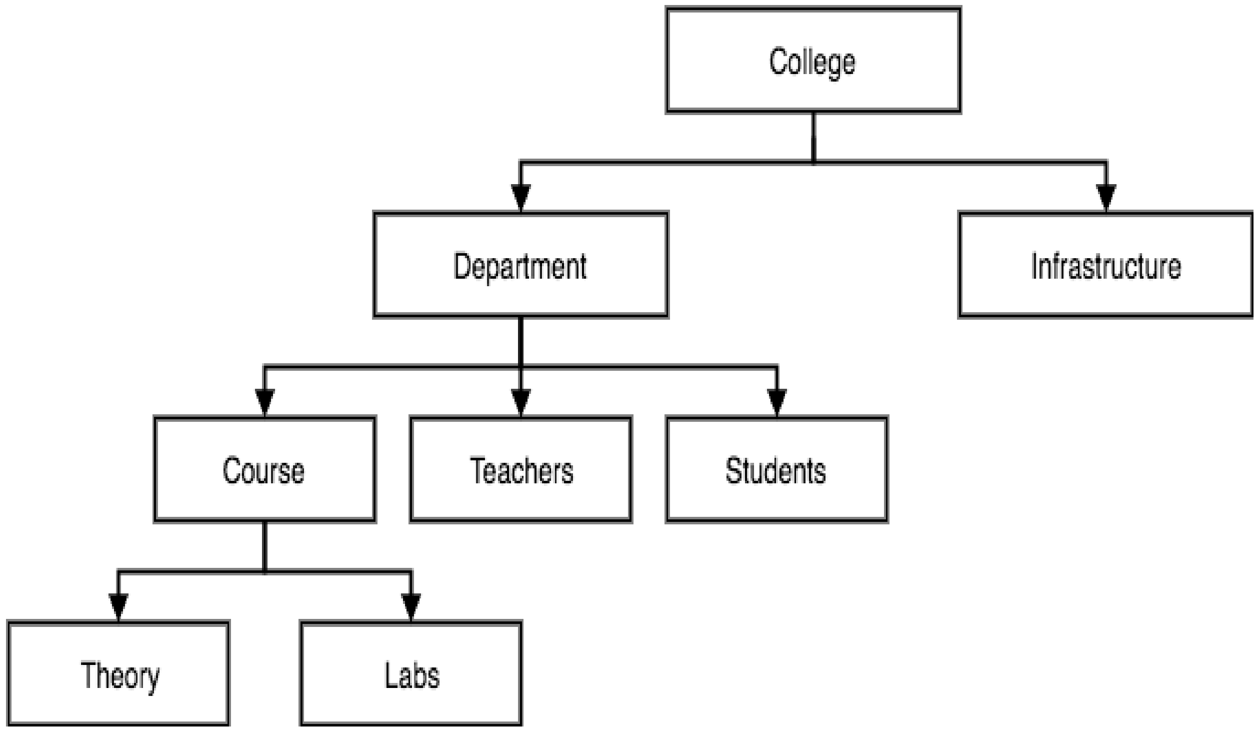
Hierarchical Model

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

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

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

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

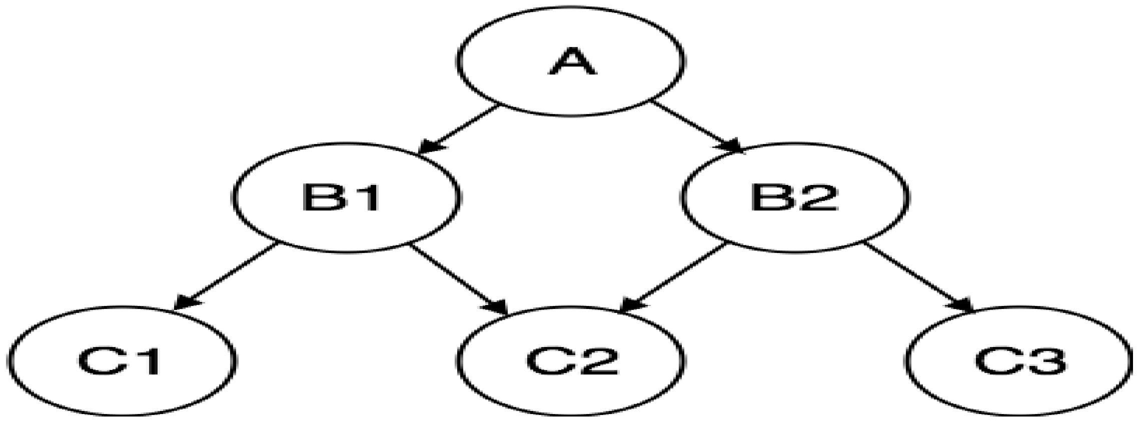


Network Model

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

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

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



Entity-relationship Model

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

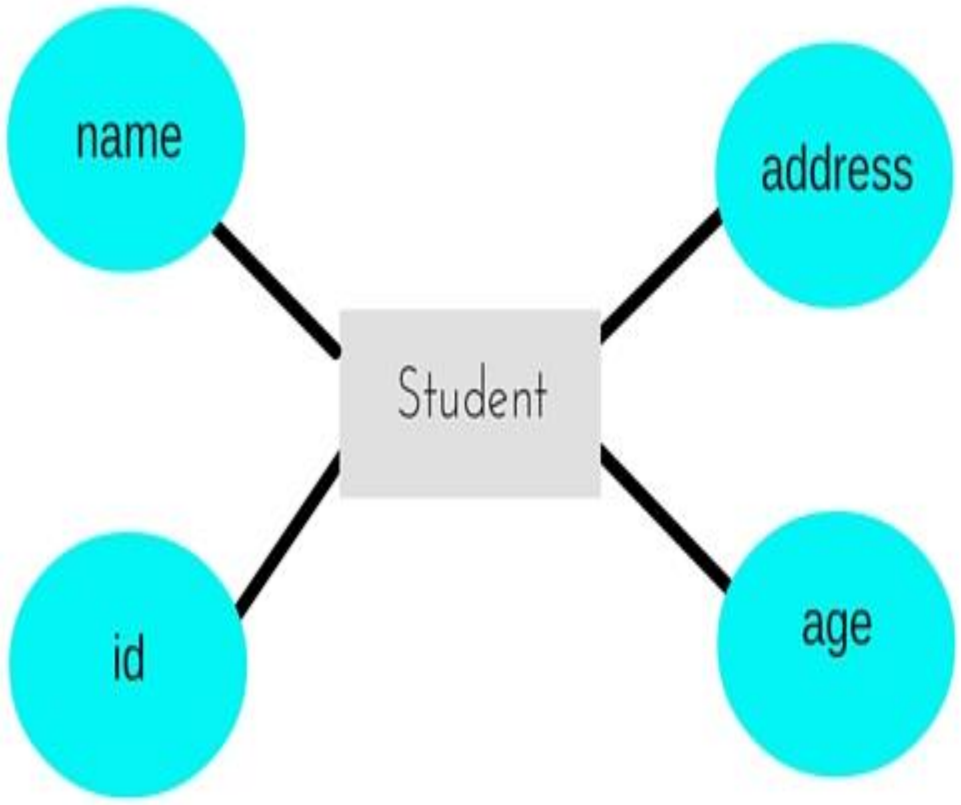
Different entities are related using relationships.

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

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

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

Relationships can also be of different types. To learn about E-R Diagrams in details, click on the link.



Relational Model

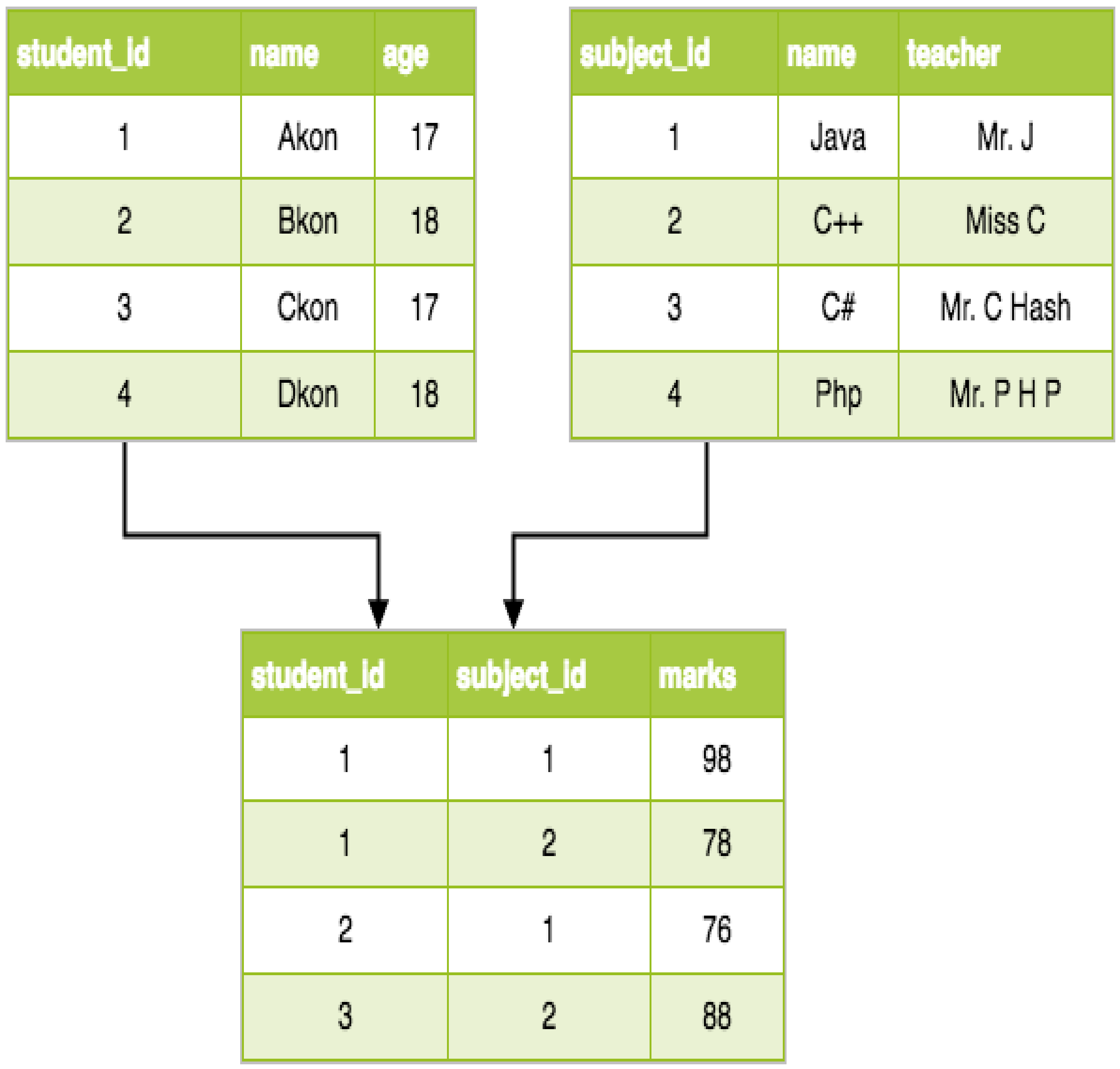
In this model, data is organised in two-dimensional **tables** and the relationship is maintained by storing a common field.

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

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

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

In the coming tutorials we will learn how to design tables, normalize them to reduce data redundancy and how to use Structured Query language to access data from tables.



student_Id	name	age
1	Akon	17
2	Bkon	18
3	Ckon	17
4	Dkon	18

subject_Id	name	teacher
1	Java	Mr. J
2	C++	Miss C
3	C#	Mr. C Hash
4	Php	Mr. P H P

student_Id	subject_Id	marks
1	1	98
1	2	78
2	1	76
3	2	88

Basic Concepts of ER Model in DBMS

As we described in the tutorial Database models, Entity-relationship model is a model used for design and representation of relationships between data.

The main data objects are termed as Entities, with their details defined as attributes, some of these attributes are important and are used to identity the entity, and different entities are related using relationships.

In short, to understand about the ER Model, we must understand about:

- Entity and Entity Set
- What are Attributes? And Types of Attributes.
- Keys
- Relationships

Let's take an example to explain everything. For a **School Management Software**, we will have to store **Student** information, **Teacher** information, **Classes**, **Subjects** taught in each class etc.

ER Model: Entity and Entity Set

Considering the above example, **Student** is an entity, **Teacher** is an entity, similarly, **Class**, **Subject** etc are also entities.

An Entity is generally a real-world object which has characteristics and holds relationships in a DBMS.

If a Student is an Entity, then the complete dataset of all the students will be the **Entity Set**

ER Model: Attributes

If a Student is an Entity, then student's **roll no.**, student's **name**, student's **age**, student's **gender** etc will be its attributes.

An attribute can be of many types, here are different types of attributes defined in ER database model:

1. **Simple attribute:** The attributes with values that are atomic and cannot be broken down further are simple attributes. For example, student's **age**.

2. **Composite attribute:** A composite attribute is made up of more than one simple attribute. For example, student's **address** will contain, **house no.**, **street name**, **pincode** etc.

3. **Derived attribute:** These are the attributes which are not present in the whole database management system, but are derived using other attributes. For example, *average age of students in a class*.

4. **Single-valued attribute:** As the name suggests, they have a single value.

5. **Multi-valued attribute:** And, they can have multiple values.

ER Model: Keys

If the attribute **roll no.** can uniquely identify a student entity, amongst all the students, then the attribute **roll no.** will be said to be a key.

Following are the types of Keys:

1. Super Key
2. Candidate Key
3. Primary Key

We have covered Keys in details here in Database Keys tutorial.

ER Model: Relationships

When an Entity is related to another Entity, they are said to have a relationship. For example, A **Class** Entity is related to **Student** entity, becasue students study in classes, hence this is a relationship.

Depending upon the number of entities involved, a **degree** is assigned to relationships.

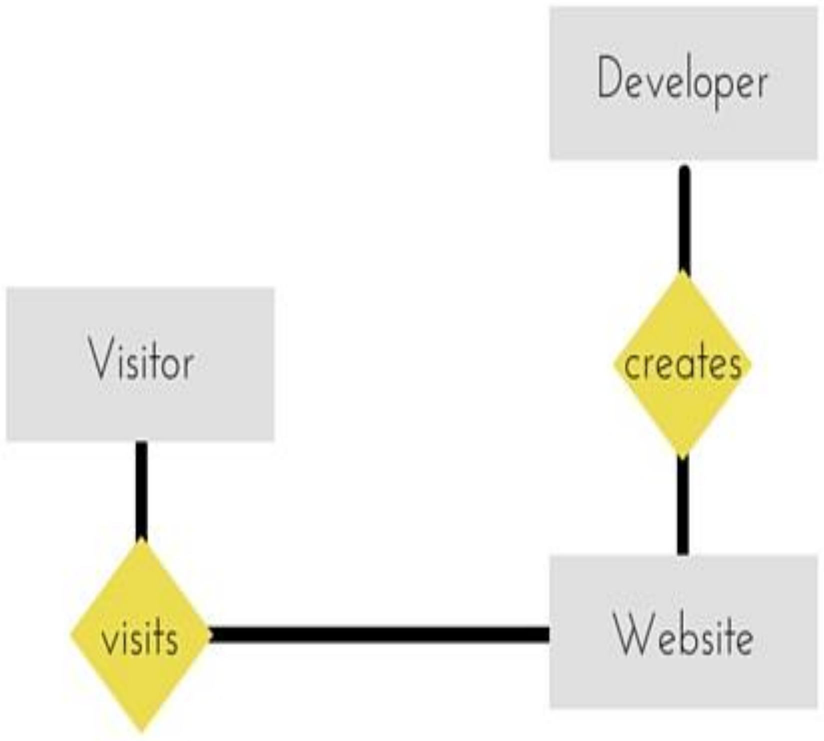
For example, if 2 entities are involved, it is said to be **Binary relationship**, if 3 entities are involved, it is said to be **Ternary** relationship, and so on.

In the next tutorial, we will learn how to create ER diagrams and design databases using ER diagrams.

Working with ER Diagrams

ER Diagram is a visual representation of data that describes how data is related to each other. In ER Model, we disintegrate data into entities, attributes and setup relationships between entities, all this can be represented visually using the ER diagram.

For example, in the below diagram, anyone can see and understand what the diagram wants to convey: *Developer develops a website, whereas a Visitor visits a website*.



Components of ER Diagram

Entitiy, Attributes, Relationships etc form the components of ER Diagram and there are defined symbols and shapes to represent each one of them.

Let's see how we can represent these in our ER Diagram.

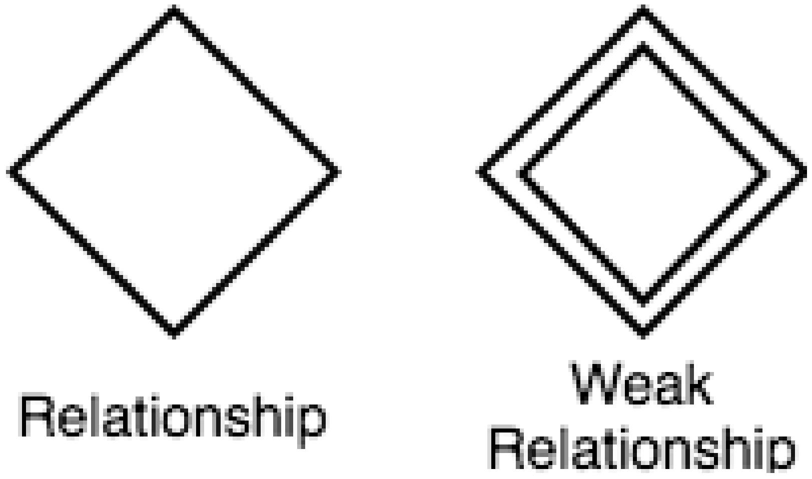
Entity

Simple rectangular box represents an Entity.

Student
Subject

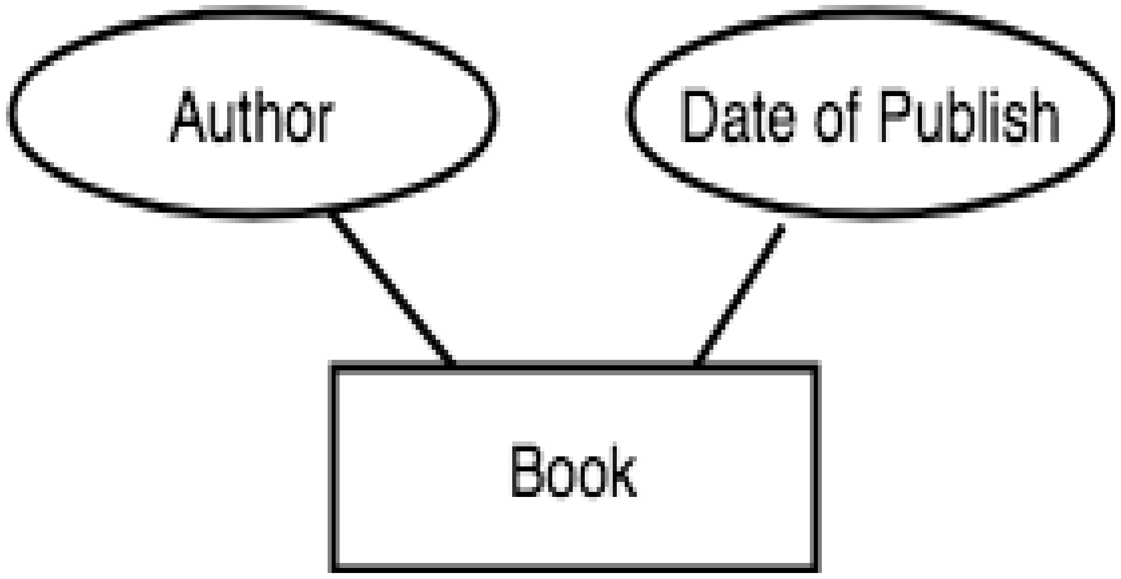
Relationships between Entities - Weak and Strong

Rhombus is used to setup relationships between two or more entities.



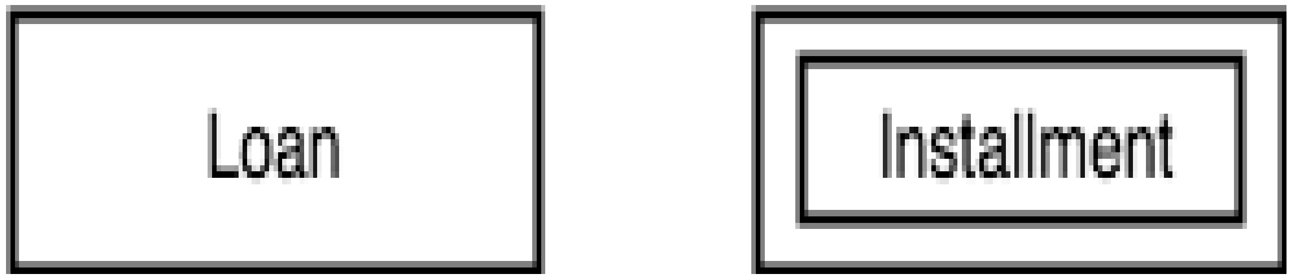
Attributes for any Entity

Ellipse is used to represent attributes of any entity. It is connected to the entity.



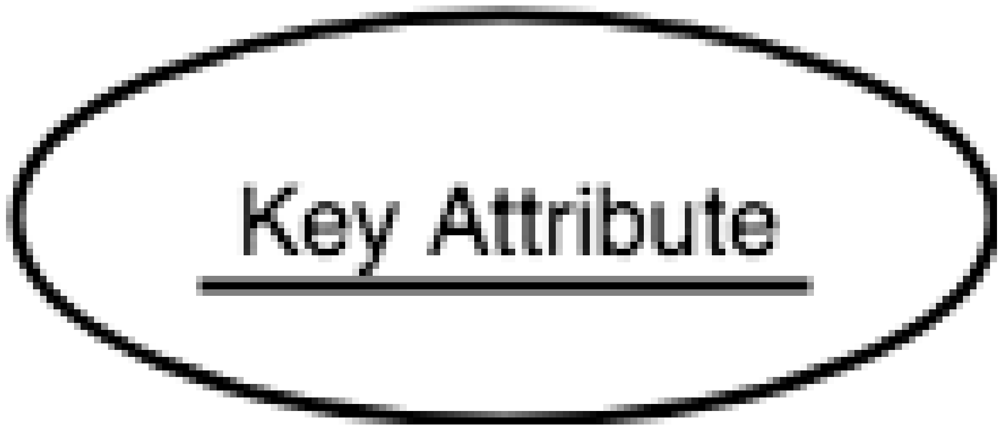
Weak Entity

A weak Entity is represented using double rectangular boxes. It is generally connected to another entity.



Key Attribute for any Entity

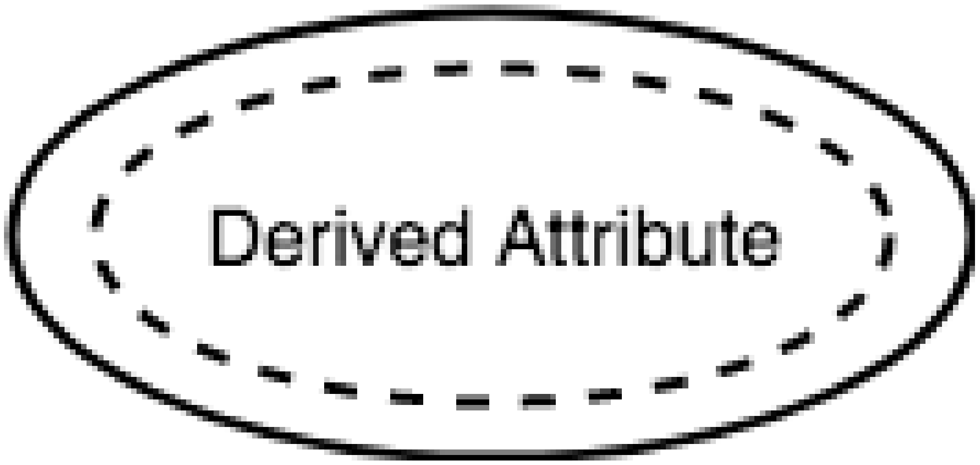
To represent a Key attribute, the attribute name inside the Ellipse is underlined.



Derived Attribute for any Entity

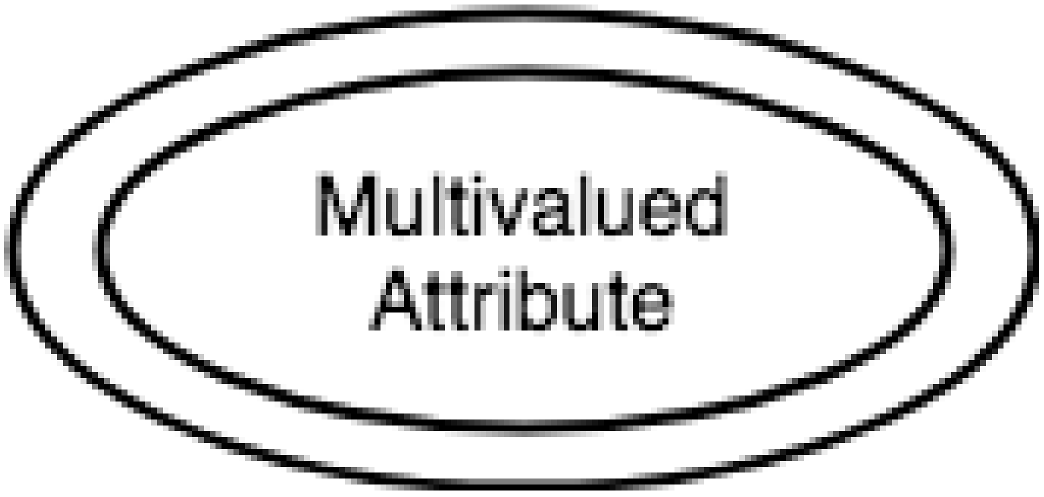
Derived attributes are those which are derived based on other attributes, for example, age can be derived from date of birth.

To represent a derived attribute, another dotted ellipse is created inside the main ellipse.



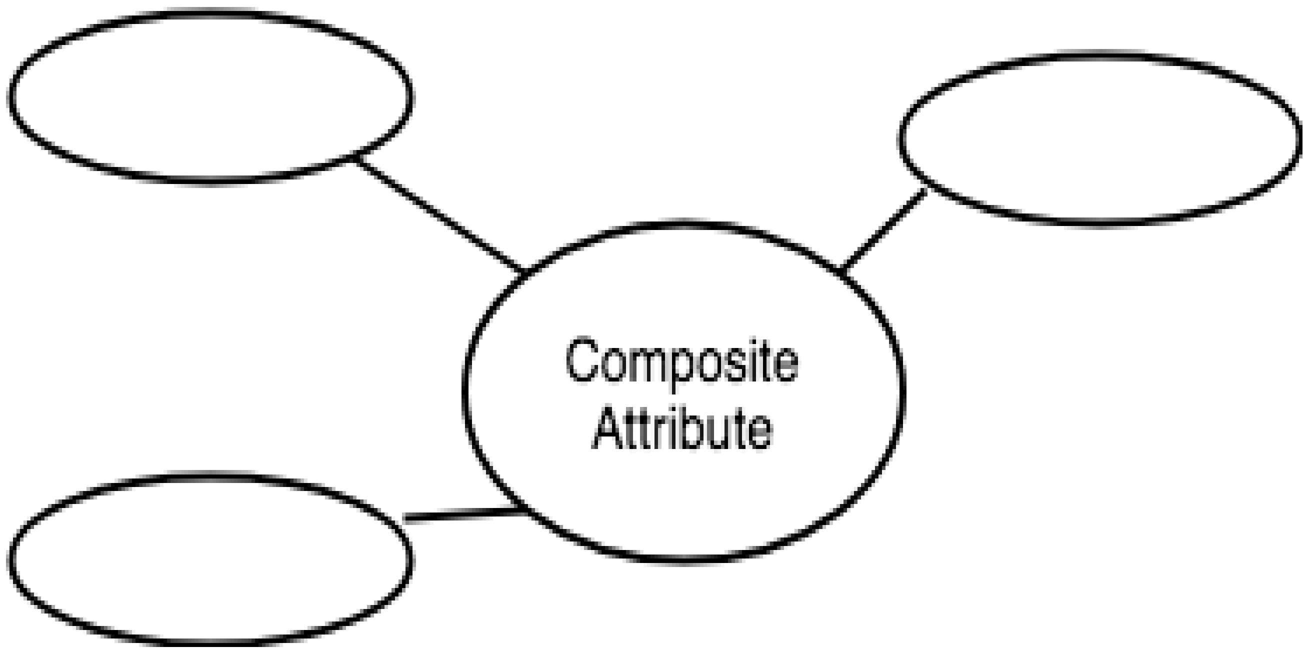
Multivalued Attribute for any Entity

Double Ellipse, one inside another, represents the attribute which can have multiple values.



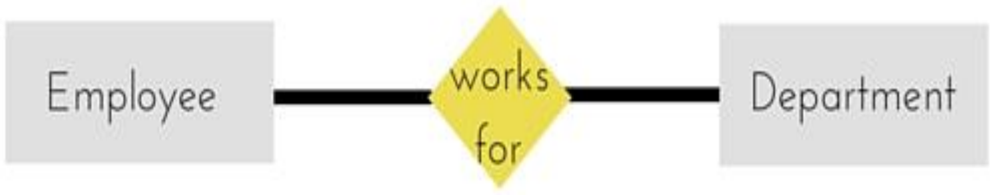
Composite Attribute for any Entity

A composite attribute is the attribute, which also has attributes.



ER Diagram: Entity

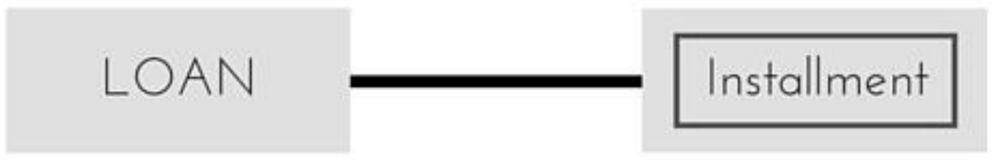
An **Entity** can be any object, place, person or class. In ER Diagram, an **entity** is represented using rectangles. Consider an example of an Organisation- Employee, Manager, Department, Product and many more can be taken as entities in an Organisation.



The yellow rhombus in between represents a relationship.

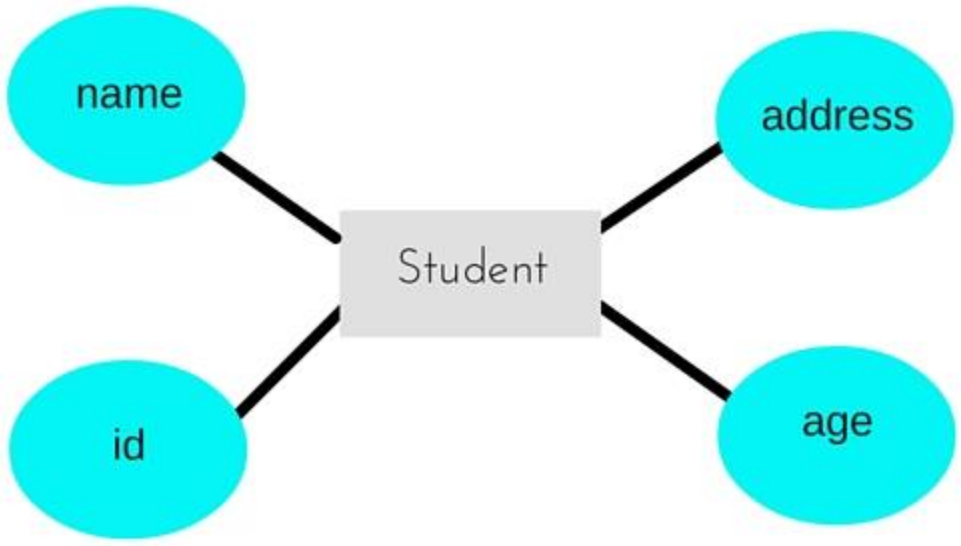
ER Diagram: Weak Entity

Weak entity is an entity that depends on another entity. Weak entity doesn't have anay key attribute of its own. Double rectangle is used to represent a weak entity.



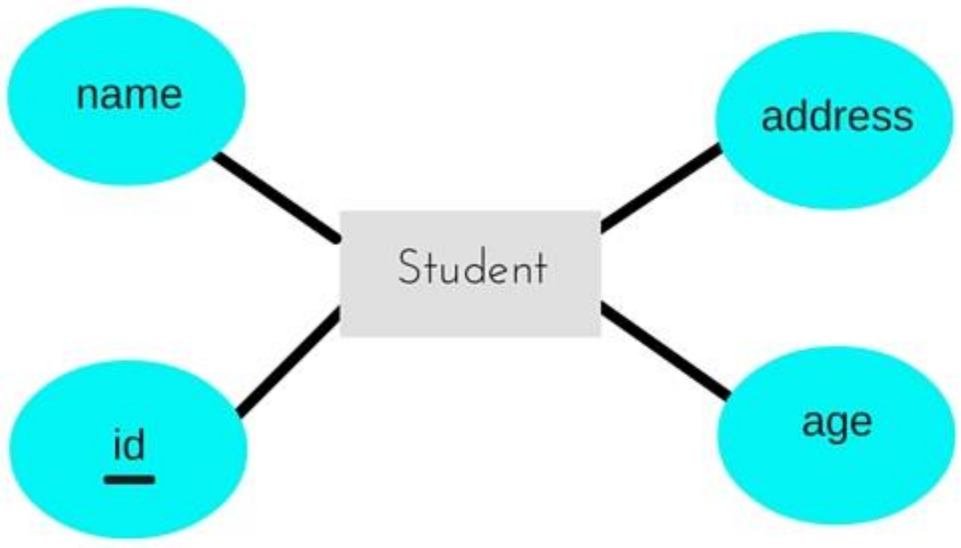
ER Diagram: Attribute

An Attribute describes a property or characterstic of an entity. For example, Name, Age, Address etc can be attributes of a Student. An attribute is represented using eclipse.



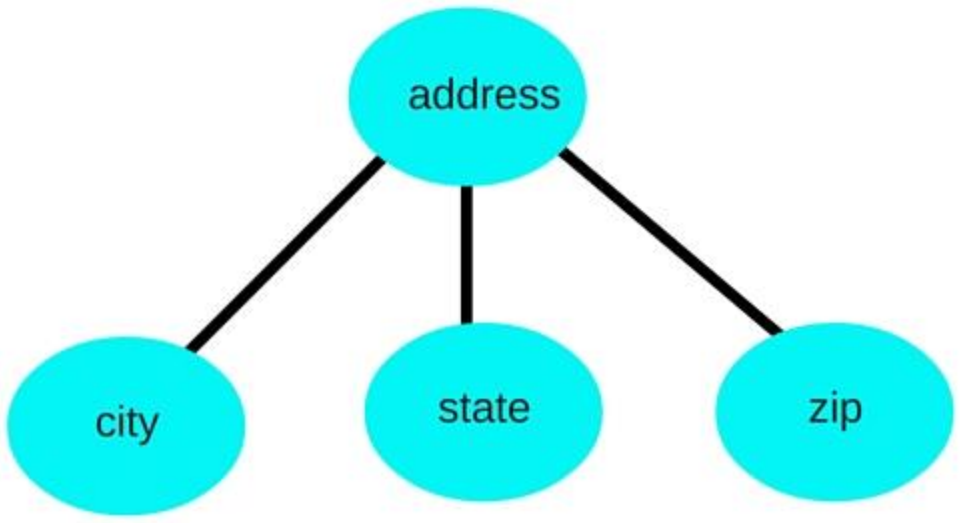
ER Diagram: Key Attribute

Key attribute represents the main characterstic of an Entity. It is used to represent a Primary key. Ellipse with the text underlined, represents Key Attribute.



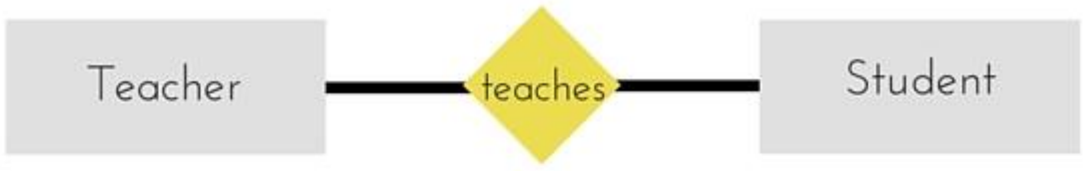
ER Diagram: Composite Attribute

An attribute can also have their own attributes. These attributes are known as **Composite** attributes.



ER Diagram: Relationship

A Relationship describes relation between **entities**. Relationship is represented using diamonds or rhombus.



There are three types of relationship that exist between Entities.

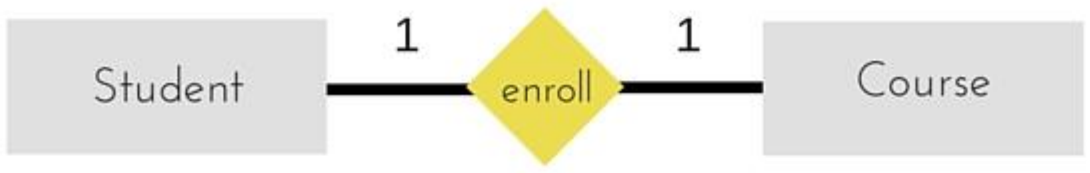
1. Binary Relationship
2. Recursive Relationship
3. Ternary Relationship

ER Diagram: Binary Relationship

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

One to One Relationship

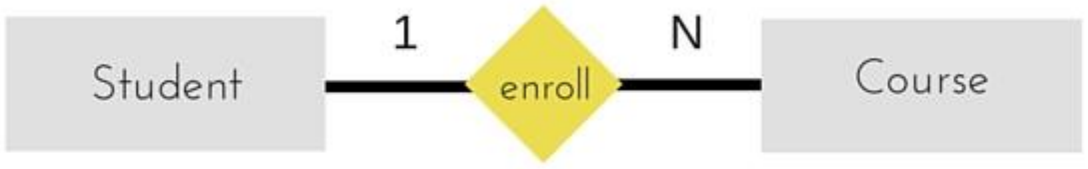
This type of relationship is rarely seen in real world.



The above example describes that one student can enroll only for one course and a course will also have only one Student. This is not what you will usually see in real-world relationships.

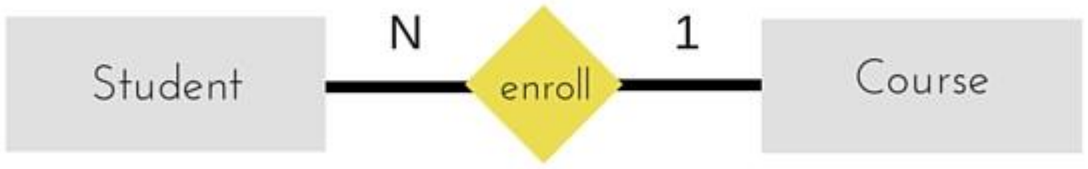
One to Many Relationship

The below example showcases this relationship, which means that 1 student can opt for many courses, but a course can only have 1 student. Sounds weird! This is how it is.

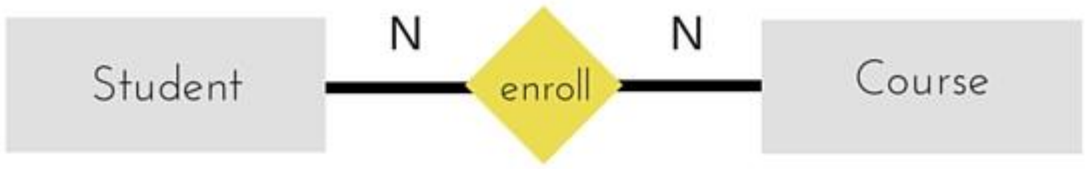


Many to One Relationship

It reflects business rule that many entities can be associated with just one entity. For example, Student enrolls for only one Course but a Course can have many Students.



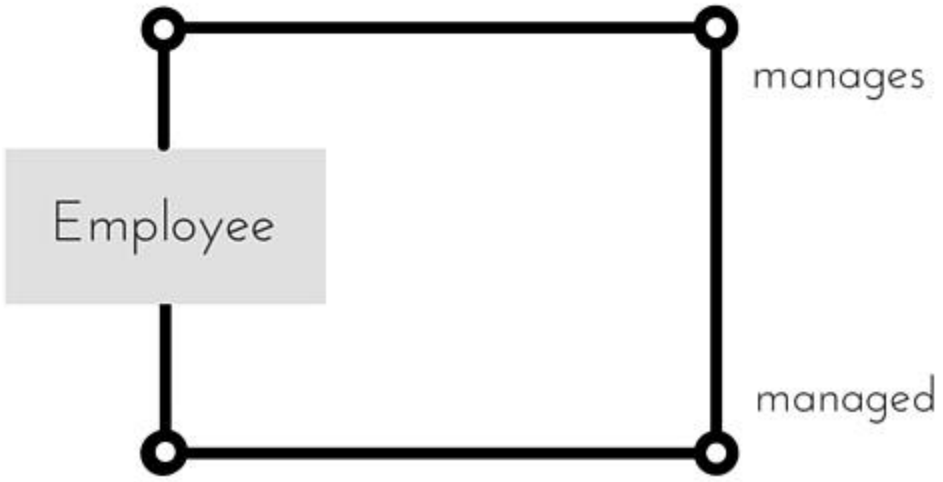
Many to Many Relationship



The above diagram represents that one student can enroll for more than one courses. And a course can have more than 1 student enrolled in it.

ER Diagram: Recursive Relationship

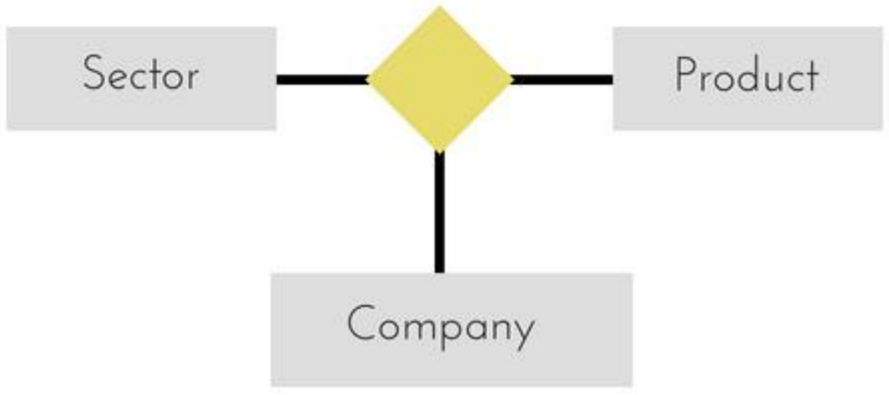
When an Entity is related with itself it is known as **Recursive** Relationship.



ER Diagram: Ternary Relationship

Relationship of degree three is called Ternary relationship.

A Ternary relationship involves three entities. In such relationships we always consider two entites together and then look upon the third.



- The above relationship involves 3 entities.
- Company operates in Sector, producing some Products.

For example, in the diagram above, we have three related entities, **Company**, **Product** and **Sector**. To understand the relationship better or to define rules around the model, we should relate two entities and then derive the third one.

A **Company** produces many **Products**/ each product is produced by exactly one company.

A **Company** operates in only one **Sector** / each sector has many companies operating in it.

Considering the above two rules or relationships, we see that although the complete relationship involves three entities, but we are looking at two entities at a time.

Advantage of DBMS (including Data independence)

DBMS A **database management system** is the software system that allows users to define, create and maintain a database and provides controlled access to the data.

A Database Management System (DBMS) is basically a collection of programs that enables users to store, modify, and extract information from a database as per the requirements. DBMS is an intermediate layer between programs and the data. Programs access the DBMS, which then accesses the data. There are different types of DBMS ranging from small systems that run on personal computers to huge systems that run on mainframes. The following are main examples of database applications:

• Computerized library systems

• Automated teller machines

• Flight reservation systems

• Computerized parts inventory systems

A database management system is a piece of software that provides services for accessing a database, while maintaining all the required features of the data. Commercially available Database management systems in the market are dbase, FoxPro, IMS and Oracle, MySQL, SQL Servers and DB2 etc.

hese systems allow users to create update, and extract information from their databases.

Compared to a manual filing system, the biggest advantages to a computerized database system are speed, accuracy, and' accessibility.

Who makes this database software?

There are a lot of database software manufacturers out there and a wide range of prices, sizes, speeds and functionalities. At the lower end of the scale are personal database software products like Microsoft Access, which is designed to be used by individuals or small companies relatively little data. User friendliness and ease of use are the priority rather than speed and scalability (in other words, it works well when you have 100 records but not when you have 100,000). At the higher end are full-fledged enterprise solutions, such as Oracle Enterprise Edition. These database software products can handle millions of data entries and are fast and efficient. They have ·many optimization and performance tools and generally require a Database Administrator (DBA) to look after them. Products in this range can also be very expensive.

In the middle are products like Microsoft SQL Server, which is a logical upgrade from Microsoft Access for Windows users. There are also several very good free database software products, such as MySQL and PostgreSQL. These are lacking on the user interface side, but can certainly compete on speed and scalability.

Developments and Evolution of DBMS Concept

We have already seen that the predecessor to the DBMS was the file-based system. However, there was never a time when the database approach began and the file-based system ceased. In fact, the file-based system still exists in specific areas. It has been suggested that the DBMS has its roots in the 1960s Apollo moon-landing project, which was initiated in response to USA's President Kennedy's objective of landing a man on the moon by the end of that decade. At that time there was no system available that would be able to handle and manage the vast amounts of information that the project would generate. As a result, North American Aviation (NAA, now Rockwell International), the prime contractor for the project, developed software known as GUAM (Generalized Update Access Method). GUAM was based on the concept that smaller components come together as parts of larger components, and so on, until the final product is assembled. This structure, which confirms to an upside down tree, is also known as a hierarchical structure.

In the mid 1960s, IBM joined NAA to develop GUAM into what is now known as IMS (Information Management System). The reason why IBM restricted IMS to the management of hierarchies of records was to allow the use of serial storage devices, most notably magnetic tape, which was a market requirement at that time. This restriction was subsequently dropped. Although one of the earliest commercial DBMS, IMS is still main hierarchical DBMS used by most large mainframe installations.

In the mid-1960s, another significant development was the emergency of IDS (Integrated Data Store) from General Electric. This work was headed by one of the early pioneers of database systems, Charles Bachmann. This development led to a new type of database system known as the network DBMS, which had a profound effect on the information systems of that generation. The network database was developed partly to address the need to represent more complex data relationships that could be modeled with hierarchical structures, and partly to impose a database standard. To help establish such standards, the Conference on Data Systems Languages (CODASYL), comprising representatives of the US government and the world of business and commerce formed a List Processing Task Force in 1965, subsequently renamed the Data Base Task Group (DBTG) in 1967. The terms of reference for the DBTG were to define standard specifications for an environment that would allow database creation and data manipulation. A draft report was issued in 1969 and the first definitive report in 1971.

Although, the report, was not formally adopted by the American National Standards Institute (ANSI), a number of systems were subsequently developed following the DBTG proposal. These systems are now known as CODASYL or DBTG systems. The CODASYL and hierarchical approaches represented the first-generation of DBMSs.

In 1970 E.F. Codd of the IBM Research Laboratory produced his highly influential paper on the relational data model. This paper was very timely and addressed the disadvantages of the former approaches. Many experimental relational DBMS's were implemented there after, with the first commercial products appearing in the late 1970s and early 1980s. Of particular note is the System R project at IBM's San Jose Research Laboratory in California, which was developed during the late 1970s.This project was designed to prove the practicality of the relational model by providing an implementation of its data structures and operations, and led to two major developments:

• The development 'of a structure query language called SQL, which has since become the standard language for relational DBMS's.

• The production of various commercial relational DBMS products during the 1980s, for example DB2 and SQL/DS from IBM and Oracle Corporation.

Now there are several hundred relational DBMSs for both mainframe and PC environments, though many are stretching the definition of the relational model. Other examples of multi-user relational DBMSs are INGRES-II from Computer Associates, and Informix Software, Inc. Examples of PC-based relational l)BMSs are Access and FoxPro from Microsoft, Paradox from Corel Corporation, InterBase and BDE from Borland, and R:Base from R:Base Technologies. Relational DBl\1Ss are referred to as second generation DBMSs

The relational model is not without its failures, and in particular its limited modeling capabilities. There has been much research since then attempting to address this problem. In 1976, Chen presented the Entity-Relationship model, which are now a widely accepted technique for database design and the basis for the methodology.

In 1979, Codd himself attempted to address some of the failures in-his original work with an extended version of the relational model called RM/T (1979) and subsequently RM/V2 (1990).The attempts to provide a data model that represents the 'real world' more closely have been loosely classified as semantic data modeling.

In response to the increasing complexity of database applications, two new systems have emerged: the Object Oriented DBMS (OODBMS) and the Object-Relational DBMS (ORDBMS). This evolution represents third-generation DBMSs.

Components of the Database System Environment

There are five major components in the database system environment and their interrelationship are.

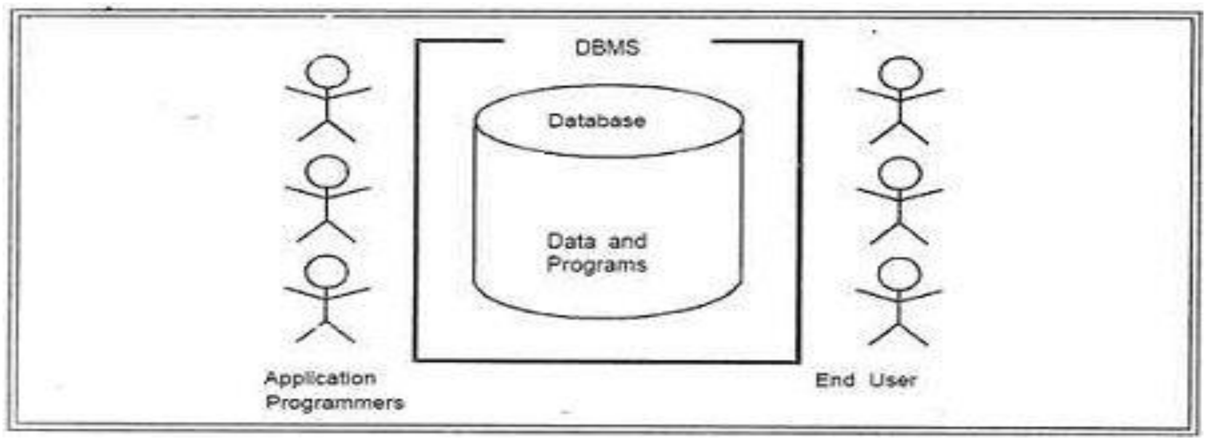
• Hardware

• Software

• Data

• Users

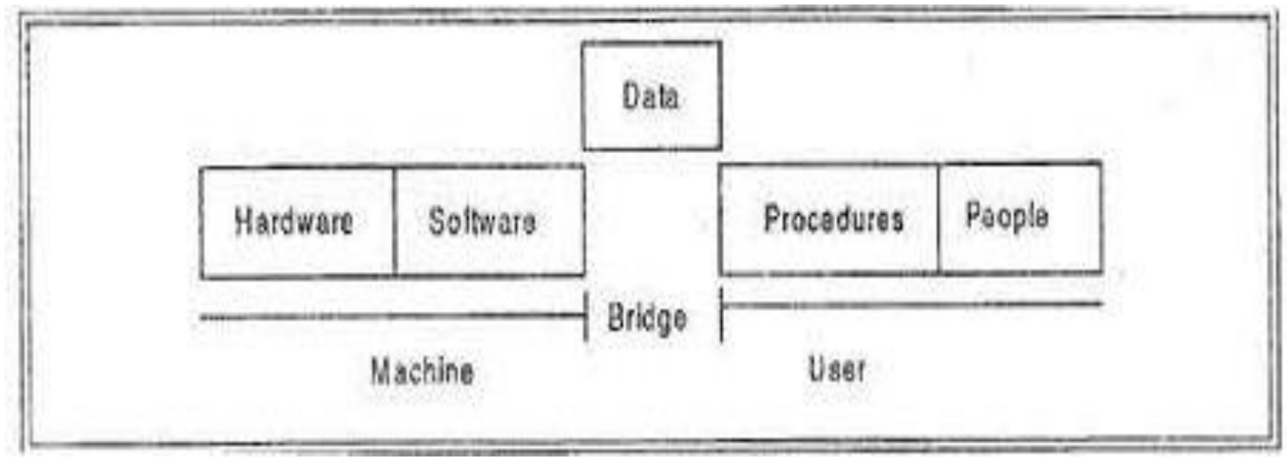
• Procedures



1.Hardware: The hardware is the actual computer system used for keeping and accessing the database. Conventional DBMS hardware consists of secondary storage devices, usually hard disks, on which the database physically resides, together with the associated Input-Output devices, device controllers and· so forth. Databases run on a' range of machines, from Microcomputers to large mainframes. Other hardware issues for a DBMS includes database machines, which is hardware designed specifically to support a database system.

2. Software: The software is the actual DBMS. Between the physical database itself (i.e. the data as actually stored) and the users of the system is a layer of software, usually called the Database Management System or DBMS. All requests from users for access to the database are handled by the DBMS. One general function provided by the DBMS is thus the shielding of database users from complex hardware-level detail.

The DBMS allows the users to communicate with the database. In a sense, it is the mediator between the database and the users. The DBMS controls the access and helps to maintain the consistency of the data. Utilities are usually included as part of the DBMS. Some of the most common utilities are report writers and application development.



3. Data : It is the most important component of DBMS environment from the end users point of view. As shown in observes that data acts as a bridge between the machine components and the user components. The database contains the operational data and the meta-data, the ‘data about data’.

The database should contain all the data needed by the organization. One of the major features of databases is that the actual data are separated from the programs that use the data. A database should always be designed, built and populated for a particular audience and for a specific purpose.

4. Users : There are a number of users who can access or retrieve data on demand using the applications and interfaces provided by the DBMS. Each type of user needs different software capabilities. The users of a database system can be classified in the following groups, depending on their degrees of expertise or the mode of their interactions with the DBMS. The users can be:

• Naive Users

• Online Users

• Application Programmers

• Sophisticated Users

• Data Base Administrator (DBA)

Naive Users: Naive Users are those users who need not be aware of the presence of the database system or any other system supporting their usage. Naive users are end users of the database who work through a menu driven application program, where the type and range of response is always indicated to the user.

A user of an Automatic Teller Machine (ATM) falls in this category. The user is instructed through each step of a transaction. He or she then responds by pressing a coded key or entering a numeric value. The operations that can be performed by valve users are very limited and affect only a precise portion of the database. For example, in the case of the user of the Automatic Teller Machine, user’s action affects only one or more of his/her own accounts.

Online Users : Online users are those who may communicate with the database directly via an online terminal or indirectly via a user interface and application program. These users are aware of the presence of the database system and may have acquired a certain amount of expertise with in the limited interaction permitted with a database.

Sophisticated Users : Such users interact with the system without ,writing programs.

Instead, they form their requests in database query language. Each such query is submitted to a very processor whose function is to breakdown DML statement into instructions that the storage manager understands.

Specialized Users : Such users are those ,who write specialized database application that do not fit into the fractional data-processing framework. For example: Computer-aided design systems, knowledge base and expert system, systems that store data with complex data types (for example, graphics data and audio data).

Application Programmers : Professional programmers are those who are responsible for developing application programs or user interface. The application programs could be written using general purpose programming language or the commands available to manipulate a database.

Database Administrator: The database administrator (DBA) is the person or group in charge for implementing the database system ,within an organization. The “DBA has all the system privileges allowed by the DBMS and can assign (grant) and remove (revoke) levels of access (privileges) to and from other users. DBA is also responsible for the evaluation, selection and implementation of DBMS package.

5. Procedures: Procedures refer to the instructions and rules that govern the design and use of the database. The users of the system and the staff that manage the database require documented procedures on how to use or run the system.

These may consist of instructions on how to:

• Log on to the DBMS.

• Use a particular DBMS facility or application program.

• Start and stop the DBMS.

• Make backup copies of the database.

• Handle hardware or software failures.

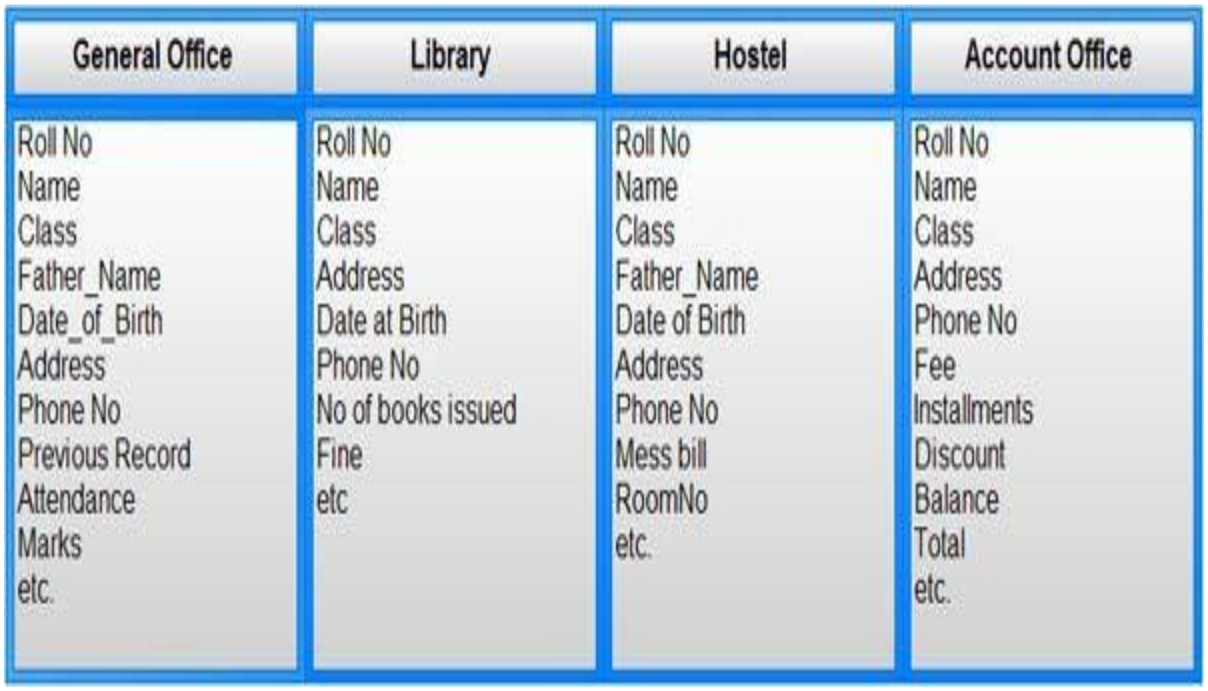
Change the structure of a table, reorganize the database across multiple disks, improve performance, or archive data to secondary storage.

Advantages of DBMS

The database management system has promising potential advantages, which are explained below:

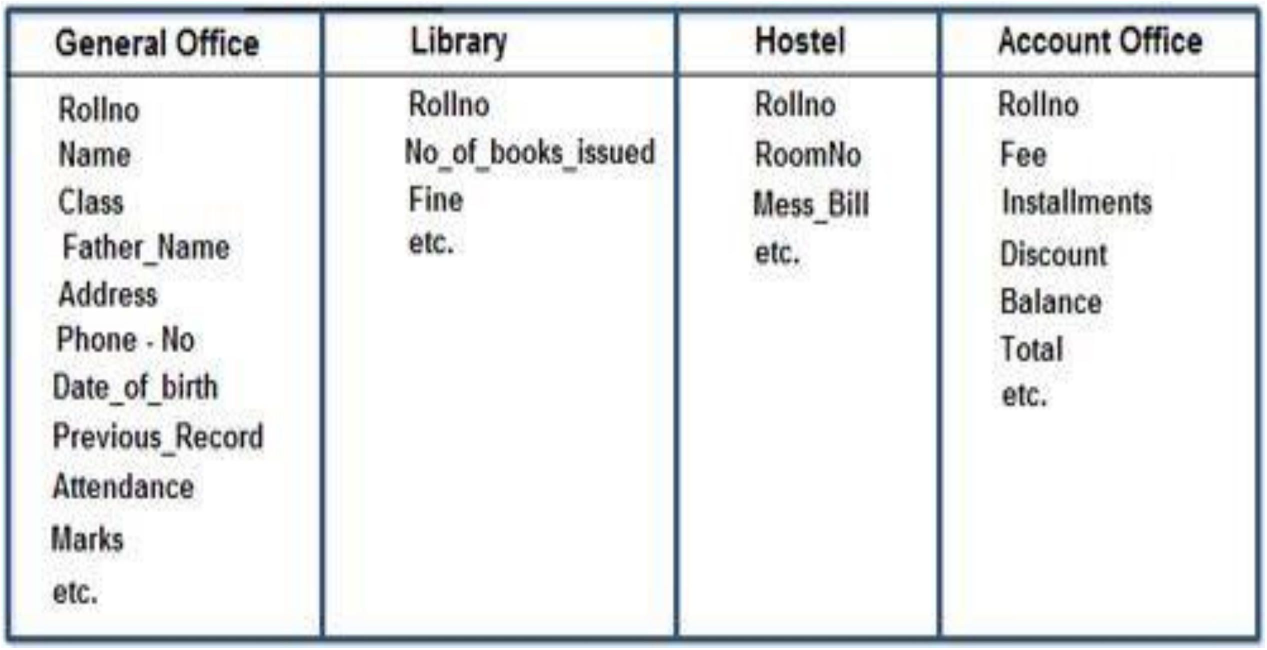
1. Controlling Redundancy: In file system, each application has its own private files, which cannot be shared between multiple applications. 1:his can often lead to considerable redundancy in the stored data, which results in wastage of storage space. By having centralized database most of this can be avoided. It is not possible that all redundancy should be eliminated. Sometimes there are sound business and technical reasons for· maintaining multiple copies of the same data. In a database system, however this redundancy can be controlled.

For example: In case of college database, there may be the number of applications like General Office, Library, Account Office, Hostel etc. Each of these applications may maintain the following information into own private file applications:



General Office	Library	Hostel	Account Office
Roll No	Roll No	Roll No	Roll No
Name	Name	Name	Name
Class	Class	Class	Class
Father_Name	Address	Father_Name	Address
Date_of_Birth	Date at Birth	Date of Birth	Phone No
Address	Phone No	Address	Fee
Phone No	No of books issued	Phone No	Installments
Previous Record	Fine	Mess bill	Discount
Attendance	etc	RoomNo	Balance
Marks		etc.	Total
etc.			etc.

It is clear from the above file systems, that there is some common data of the student which has to be mentioned in each application, like Rollno, Name, Class, Phone_No~ Address etc. This will cause the problem of redundancy which results in wastage of storage space and difficult to maintain, but in case of centralized database, data can be shared by number of applications and the whole college can maintain its computerized data with the following database:



General Office	Library	Hostel	Account Office
Rollno	Rollno	Rollno	Rollno
Name	No_of_books_issued	RoomNo	Fee
Class	Fine	Mess_Bill	Installments
Father_Name	etc.	etc.	Discount
Address			Balance
Phone - No			Total
Date_of_birth			etc.
Previous_Record			
Attendance			
Marks			
etc.			

It is clear in the above database that Rollno, Name, Class, Father_Name, Address,

Phone_No, Date_of_birth which are stored repeatedly in file system in each application, need not be stored repeatedly in case of database, because every other application can access this information by joining of relations on the basis of common column i.e. Rollno. Suppose any user of Library system need the Name, Address of any particular student and by joining of Library and General Office relations on the basis of column Rollno he/she can easily retrieve this information.

Thus, we can say that centralized system of DBMS reduces the redundancy of data to great extent but cannot eliminate the redundancy because RollNo is still repeated in all the relations.

2. Integrity can be enforced: Integrity of data means that data in database is always accurate, such that incorrect information cannot be stored in database. In order to maintain the integrity of data, some integrity constraints are enforced on the database. A DBMS should provide capabilities for defining and enforcing the constraints.

For Example: Let us consider the case of college database and suppose that college having only BTech, MTech, MSc, BCA, BBA and BCOM classes. But if a \.,ser enters the class MCA, then this incorrect information must not be stored in database and must be prompted that this is an invalid data entry. In order to enforce this, the integrity constraint must be applied to the class attribute of the student entity. But, in case of file system tins constraint must be enforced on all the application separately (because all applications have a class field).

In case of DBMS, this integrity constraint is applied only once on the class field of the

General Office (because class field appears only once in the whole database), and all other applications will get the class information about the student from the General Office table so the integrity constraint is applied to the whole database. So, we can conclude that integrity constraint can be easily enforced in centralized DBMS system as compared to file system.

3. Inconsistency can be avoided : When the same data is duplicated and changes are made at one site, which is not propagated to the other site, it gives rise to inconsistency and the two entries regarding the same data will not agree. At such times the data is said to be inconsistent. So, if the redundancy is removed chances of having inconsistent data is also removed.

Let us again, consider the college system and suppose that in case of General_Office file

it is indicated that Roll_Number 5 lives in Amritsar but in library file it is indicated that

Roll_Number 5 lives in Jalandhar. Then, this is a state at which tIle two entries of the same object do not agree with each other (that is one is updated and other is not). At such time the database is said to be inconsistent.

An inconsistent database is capable of supplying incorrect or conflicting information. So there should be no inconsistency in database. It can be clearly shown that inconsistency can be avoided in centralized system very well as compared to file system ..

Let us consider again, the example of college system and suppose that RollNo 5 is .shifted from Amritsar to Jalandhar, then address information of Roll Number 5 must be updated, whenever Roll number and address occurs in the system. In case of file system, the information must be updated separately in each application, but if we make updation only at three places and forget to make updation at fourth application, then the whole system show the inconsistent results about Roll Number 5.

In case of DBMS, Roll number and address occurs together only single time in General_Office table. So, it needs single updation and then an other application retrieve the address information from General_Office which is updated so, all application will get the current and latest information by providing single update operation and this single update operation is propagated to the whole database or all other application automatically, this property is called as Propagation of Update.

We can say the redundancy of data greatly affect the consistency of data. If redundancy is less, it is easy to implement consistency of data. Thus, DBMS system can avoid inconsistency to great extent.

4. Data can be shared: As explained earlier, the data about Name, Class, Father name etc. of General Office is shared by multiple applications in centralized DBMS as compared to file system so now applications can be developed to operate against the same stored data. The applications may be developed without having to create any new stored files.

5. Standards can be enforced : Since DBMS is a central system, so standard can be enforced easily may be at Company level, Department level, National level or International level. The standardized data is very helpful during migration or interchanging of data. The file system is an independent system so standard cannot be easily enforced on multiple independent applications.

6. Restricting unauthorized access: When multiple users share a database, it is likely that some users will not be authorized to access all information in the database. For example, account office data is often considered confidential, and hence only authorized persons are allowed to access such data. In addition, some users may be permitted only to retrieve data, whereas other are allowed both to retrieve and to update. Hence, the type of access operation retrieval or update must also be controlled. Typically, users or user groups are given account numbers protected by passwords, which they can use to gain access to the database. A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and to specify account restrictions. The DBMS should then enforce these restrictions automatically.

7. Solving Enterprise Requirement than Individual Requirement: Since many types of users with varying level of technical knowledge use a database, a DBMS should provide a variety of user interface. The overall requirements of the enterprise are more important than the individual user requirements. So, the DBA can structure the database system to provide an overall service that is “best for the enterprise”.

For example: A representation can be chosen for the data in storage that gives fast access for the most important application at the cost of poor performance in some other application. But, the file system favors the individual requirements than the enterprise requirements

8. Providing Backup and Recovery: A DBMS must provide facilities for recovering from hardware or software failures. The backup and recovery subsystem of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update program, the recovery subsystem is responsible for making sure that the .database is restored to the state it was in before the program started executing.

9. Cost of developing and maintaining system is lower: It is much easier to respond to unanticipated requests when data is centralized in a database than when it is stored in a conventional file system. Although the initial cost of setting up of a database can be large, but the cost of developing and maintaining application programs to be far lower than for similar service using conventional systems. The productivity of programmers can be higher in using non-procedural languages that have been developed with DBMS than using procedural languages.

10. Data Model can be developed : The centralized system is able to represent the complex data and interfile relationships, which results better data modeling properties. The data madding properties of relational model is based on Entity and their Relationship, which is discussed in detail in chapter 4 of the book.

11. **Concurrency Control :** DBMS systems provide mechanisms to provide concurrent access of data to multiple users.

Disadvantages of DBMS

The disadvantages of the database approach are summarized as follows:

1. Complexity : The provision of the functionality that is expected of a good DBMS makes the DBMS an extremely complex piece of software. Database designers, developers, database administrators and end-users must understand this functionality to take full advantage of it. Failure to understand the system can lead to bad design decisions, which can have serious consequences for an organization.

2. Size : The complexity and breadth of functionality makes the DBMS an extremely large piece of software, occupying many megabytes of disk space and requiring substantial amounts of memory to run efficiently.

3. Performance: Typically, a File Based system is written for a specific application, such as invoicing. As result, performance is generally very good. However, the DBMS is written to be more general, to cater for many applications rather than just one. The effect is that some applications may not run as fast as they used to.

4. Higher impact of a failure: The centralization of resources increases the vulnerability of the system. Since all users and applications rely on the ~vailabi1ity of the DBMS, the failure of any component can bring operations to a halt.

5. Cost of DBMS: The cost of DBMS varies significantly, depending on the environment and functionality provided. There is also the recurrent annual maintenance cost.

6. Additional Hardware costs: The disk storage requirements for the DBMS and the database may necessitate the purchase of additional storage space. Furthermore, to achieve the required performance it may be necessary to purchase a larger machine, perhaps even a machine dedicated to running the DBMS. The procurement of additional hardware results in further expenditure.

7. Cost of Conversion: In some situations, the cost oftlle DBMS and extra hardware may be insignificant compared with the cost of converting existing applications to run on the new DBMS and hardware. This cost also includes the cost of training staff to use these new systems and possibly the employment of specialist staff to help with conversion and running of the system. This cost is one of the main reasons why some organizations feel tied to their current systems and cannot switch to modern database technology.

When not to Use a DBMS

In spite of the advantages of using a DBMS, there are a few situations in which such a system may involve unnecessary overhead costs, as that would not be incurred in traditional file processing.

The overhead costs of using a DBMS are due to the following:

- High initial investment in hardware, software, and training.
- Generality that a DBMS provides for defining and processing data.
- Overhead for providing security, concurrency control, recovery, and integrity functions.

Additional problems may arise, if the database designers and DBA do not properly design the database or if the database systems applications are not implemented properly.

Hence, it may be more desirable to use regular files under the following circumstances:

- The database and applications are simple, well defined and not expected to change.
- There are tight real-time requirements for some programs that may not be met because of DBMS overhead.
- Multiple user access to data is not required.
- An application may need to manipulate the data in a way not supported by the query language.

Requirement for a DBMS

The software responsible for the management data in computers i.e. DBMS (like Oracle, Foxpro, SQL Server etc.) should meet the following requirements:

Provide data definition facilities

It should support Data Definition Language (DDL) and provides user accessible catalog Known as Data Dictionary.

Provide facilities for storing, retrieving and updating data

It should support Data Manipulation Language (DML), so that required data can be inserted, updated, deleted and retrieved.

Supports multiple view of data

The end user should have the facility of flexible query language so that required information can be accessed easily.

Provides facilities for specifying Integrity constraints

It should support the constraints like Primary key, foreign key during creation of tables so that only the valid information is stored in the database. As soon as, we try to insert any incorrect information it should display the error message.

Provide security of data

It should have the facilities for controlling access to data and prevent unauthorized access and update.

Provide concurrency control mechanism

It should allow simultaneous access and update of data by multiple users

Support Transactions

It should support all the properties of transaction known as ACID properties. It means a sequence of operations to be performed as a whole. In other words all operations are performed or none.

Provide facilities for database recovery

It should bring database back to consistent state after a failure such as disk failure, faulty program etc.

Provide facilities for database maintenance

It should support maintenance operations like unload, reload, mass insertion, deletion and validation of data.

Master and transaction file

A master file stores relatively static data. It changes occasionally and stores all the details of the object. For example, in case of banking software the customer file which contain the data about the customer like customer_id, account_no, account_type, name, address, phone_number etc. is a master file, because it contain the static data and whole information about the customer.

The other file, which contains the data about the customer transactions, is called as a Transaction file. The customer transaction file contains the data about the account_no,\ transaction_)d, date, transaction_type (e.g. deposit or withdrawal), amount, balance etc. It is dynamic file and updated each time for any withdrawal and deposit on a given account number.

Types of database users

There are distinct types of people who engage with database management system and **End Users** are well known. However, when referring End Users, does it represent all people work with DBMS or just business users?

Theoretically, there are four types of people involve with databases and database management systems; Database Administrators, Database Designers, Application Developers and End Users. This means, we are not part of End Users group and they work with databases differently.

There are mainly two types of End Users;

Naïve users

These users have no clue on database management systems and do not know how to access database or how to make requests from database directly. Naïve users typically access the database through given GUIs that has readable menu items for opening windows forms or web pages to interact with data. Relevant example for this is, bank executive opens the interface given for entering customer information.

Sophisticated users

This group represents people who know about the structure defined in the database for some extent. They have skills for accessing the database directly (or through another interface that requires database and SQL knowledge to use) and they make direct requests to the database for getting required data down. Most smart business users and data stewards are fallen into this group. When considering modern database implementations, sophisticated users do not access the core database unless there are specific objects (like Views) created specifically for them, but they access via configured databases such as reporting databases, relational and OLAP data warehouses or data models such as Microsoft Multi-dimensional models, tabular models and models designed with Excel+Power Pivot.

Different Types of Database Users in DBMS
Application Programmers

As its name shows, application programmers are the one who writes application programs that uses the database. These application programs are written in programming languages like COBOL or PL (Programming Language 1), Java and fourth generation language. These programs meet the user requirement and made according to user requirements. Retrieving information, creating new information and changing existing information is done by these application programs.

They interact with DBMS through DML (Data manipulation language) calls. And all these functions are performed by generating a request to the DBMS. If application programmers are not there then there will be no creativity in the whole team of Database.

End Users

End users are those who access the database from the terminal end. They use the developed applications and they don’t have any knowledge about the design and working of database. These are the second class of users and their main motto is just to get their task done. There are basically two types of end users that are discussed below.

Casual User

These users have great knowledge of query language. Casual users access data by entering different queries from the terminal end. They do not write programs but they can interact with the system by writing queries.

Naïve

Any user who does not have any knowledge about database can be in this category. There task is to just use the developed application and get the desired results. For example: Clerical staff in any bank is a naïve user. They don’t have any dbms knowledge but they still use the database and perform their given task.

DBA (Database Administrator)

DBA can be a single person or it can be a group of person. Database Administrator is responsible for everything that is related to database. He makes the policies, strategies and provides technical supports.

System Analyst

System analyst is responsible for the design, structure and properties of database. All the requirements of the end users are handled by system analyst. Feasibility, economic and technical aspects of DBMS is the main concern of system analyst.
So it was all about **Different Types of Database Users in DBMS**. If you have any query regarding these users in DBMS then please comment below.

Role of Database administrator

Database administrators or managers create and maintain databases compatible with their companies’ needs. These information technology (IT) professionals oversee database updates, storage, security, and troubleshooting.

The following page provides an overview of data administration and related career paths, including descriptions of daily tasks, key skills, and salary and job prospects by industry context and location. It also outlines recommended steps for aspiring data administrators and introduces continuing education platforms, job search tools, professional organizations, and other career development resources.

What Does a Database Administrator Do?

Database administrators maintain and protect sensitive information and provide access to important datasets for companies, institutions, and government bodies. The **U.S. Bureau of Labor Statistics** (BLS) projects that database administration jobs will grow by 9% from 2018-2028. These professionals organize sensitive datasets such as financial records, purchase histories, and customer details, and they make materials available to other professionals while maintaining information security and privacy settings.

Database administrators also back up, restore, and troubleshoot database sets and system access, updating and integrating old programs to implement the latest technology.

Database administrators need at least a bachelor's degree in information science or computer science for most entry-level positions. Depending on the size and complexity of their company or governing body, these professionals may need a master's degree in database administration or information technology. All database administrators need fundamental knowledge of structured query language (SQL) and software vendor certifications.

Key Hard Skills

Like many scientific and technical professionals, database administrators need to master several hard skills to obtain and perform their jobs. The hard skills listed below require time, concentration, and technological aptitude to acquire. Keep in mind that the specific hard skills for database administration typically vary by company, position, and project.

- **SQL:** SQL, a computer language, orients and organizes all data management systems. Students should understand the three dominant database languages: Microsoft SQL, Oracle Database, and IBM's DB2. Professionals may need to build websites with MySQL, create relational connections among multiple datasets with Transact-SQL, and control object-oriented concepts with PL/SQL.

- **UNIX:** This portable, multi-user, multitasking operating system interface is made available by The Open Group. Written in C programming language, UNIX provides the organizational foundation for most Mac, Android, Chrome, and PlayStation systems. Therefore, it represents a critical building block for database management and administration.

- **Linux:** Modeled after UNIX, the open-source operating system Linux operates with versatility to control a variety of computer systems, including smartphones and supercomputers. Linux is free to install, and users can access online coding and troubleshooting hacks to create comprehensive platforms for clients and companies.

- **Oracle:** A relational database framework oriented through SQL, Oracle provides access to and organizes datasets. It also integrates information into user-friendly platforms for companies and institutions. Similar to SQL, Oracle works with most major platforms, including Windows, UNIX, Linux, and Mac OS.

- **Windows Operating System:** Windows OS represents the orienting graphic interface for all Microsoft products. The core operating system for Microsoft desktops and programs, Windows differs from Linux and UNIX systems in that it is corporately owned and not openly accessible. Currently, Windows OS is the dominant system worldwide.

- **Data Analysis:** By inspecting, collating, and interpreting datasets, data analysts translate massive bodies of information into useful and illustrative material for companies and clients. Analysts also use analytical interpretation to streamline systems by cleaning datasets, which involves prioritizing and itemizing metadata, and by providing insights on system improvements.

- **Microsoft Access:** This is the key information management tool behind referencing, reporting, and data analysis in data administration. MS Access translates metadata sets, particularly from Microsoft Excel sheets, into usable, searchable datasets. Database administrators use this tool to prioritize relationships among datasets and coalesce materials.

- **HTML:** HTML, or Hypertext Markup Language, forms the graphics of platform and website design. Professionals create interactive systems and webpages using this standard coding language, which integrates scripting languages such as JavaScript. HTML is the bedrock of computer programming, so database administrators must understand it.

Key Soft Skills

Database administrators must also possess some general, less-quantifiable strengths, and skills, often referred to as “soft skills.” Communication, organization, and problem-solving prove useful in almost any position. Companies making data-driven decisions particularly appreciate candidates with analytics and business acumen.

- **Analytics:** The ability to systematically analyze data or statistics helps database administrators identify and meet their companies’ data management needs. Database administrators often conduct updated analyses of their databases, as well.

- **Business-Focused:** Companies trust database administrators to make cost-effective decisions regarding technology and staff for data storage, maintenance, security, and analysis. Technical and business knowledge and skills make database administrators more valuable and versatile.

- **Communication:** Database administrators often supervise other IT staff, and good communication skills prove valuable to IT teamwork and leadership. Database administrators also communicate with executive management, suppliers, and technology professionals at other organizations.

- **Problem-solving:** The capacity to identify, test, and eliminate potential problems and their causes is extremely valuable to database administrators, who spend a considerable amount of time troubleshooting. Creativity also comes into play, as database administrators often need to generate new solutions to new problems.

- **Organization:** Databases require considerable organization, and database administrators organize data to make database decisions and create reports. They also organize IT department tasks and employees.

Daily Tasks

Once they have determined user needs and set up databases with appropriate disk space, network requirements, and memory, database administrators may spend their days using software tools to organize and store company records, user information, and other data. Other daily tasks include upgrading database servers and applications, modifying database structure as needed, generating user profiles, and monitoring database security.

Database Administrator Salary Information

PayScale data in the tables below suggests that data administrator salaries depend on experience level, with entry-level database administrators making a median annual salary of $53,292 while experienced professionals earn about $90,167. Entry-level professionals who earn database administration-related certifications or degrees often move up the salary scale more quickly.

Job opportunities and salaries also vary considerably by industry and location. For example, the computer systems design and related services industry employs nearly twice as many database administrators as any other industry. Meanwhile, nonresidential building construction, computer manufacturing, and oil and gas extraction pay database administrators the highest salaries, closely followed by the auto manufacturing and finance industries.

Company size can also influence salary, as larger, more complex companies often employ advanced, high-salary data professionals. Database administrators typically enjoy more career opportunities and higher salaries in major metropolitan areas. Top-paying states for database administrators include New Jersey, Washington, and California. Texas, California, and New York hire the most data administrators. California and New York land in the top five for both lists.

Database Administrators by Job Level

Entry Level (0-12 Months)	$53,292
Early Career (1-4 Years)	$62,697
Mid-Career (5-9 Years)	$78,588
Experienced (10-19 Years)	$90,167

Source: PayScale

How to Become a Database Administrator

Earn Your Degree

IT jobs rely heavily on hard skills, which candidates can build through self-study and online learning tools. Still, most positions require an IT- or CS-related bachelor’s degree, as well.

Some schools offer IT bachelor's degree programs with concentrations in database administration or management.

These programs teach students computer programming languages like Python, HTML5, CSS, and C++, through coursework on data structure, network architecture, web programming, and software applications. Many programs require completion of either an internship or a capstone project to graduate.

Database systems management and software programs vary by employer, so job-seekers with bachelor's degrees may need to earn additional certifications in specific database systems or software programs by Microsoft, IBM, Oracle, Altibase, and others.

Gain Experience

To become database administrators, job-seekers must demonstrate prior success in a related position and industry. Consequently, aspiring database administrators often benefit from bachelor's programs with IT internship programs and/or portfolio capstone projects. These avenues help students land IT positions as developers or systems administrators, and candidates demonstrating success in these roles have a better chance at database administrator positions.

Requisite hard skills vary by position, but entry-level database administrators often need skills in database administration and reporting, Oracle, IBMDB2, Altibase, SQL, and SAP Sybase ASE. Professionals seeking mid-level database administrator positions should have data analysis skills, additional certifications, and four or more years of professional experience.

Joining database administration student chapters or networking groups through professional organizations or school programs can help students obtain the information and connections necessary to land jobs in this field.

Earn Credentials

Associate and bachelor's programs concentrated on database administration include courses on programming languages, relevant software, and systems management programs. Still, some aspiring database administrators may need additional professional certifications. Candidates with general bachelor’s degrees in CS or IT, for example, may need additional database administrator certification.

Fortunately, job-seekers can pursue focused, efficient professional certifications online at varying levels of expertise tailored to their career needs and aspirations. For example, Microsoft offers entry-level, associate, and advanced SQL server certifications. Meanwhile, Oracle offers database and MySQL certifications at various levels, and IBM provides an intermediate database administrator certification for Linux, UNIX, and Windows. Certification programs typically involve several modules or courses, culminating in one or more exams.

Types of Careers in Database Administration

Database administrator degree programs equip students with foundational skills and knowledge useful in several IT and CS careers. Computer systems analysts with the right programming skills sometimes obtain jobs with only an associate degree; however, most network architects, computer programmers, and software developers need bachelor’s degrees plus related certifications and work experience. Professionals in managerial roles at large companies may need master’s degrees, as well.

Job opportunities and salaries also differ considerably by industry and location, with the bulk of jobs appearing in the computer systems design and related services industry and near major metropolitan areas. BLS data shows that database administrators earned a median annual salary of $90,070 in 2018. Higher education and certification typically boost salary potential in this field.

UNIT - II

Data Models

Brief overview of Hierarchical and Network Model :-

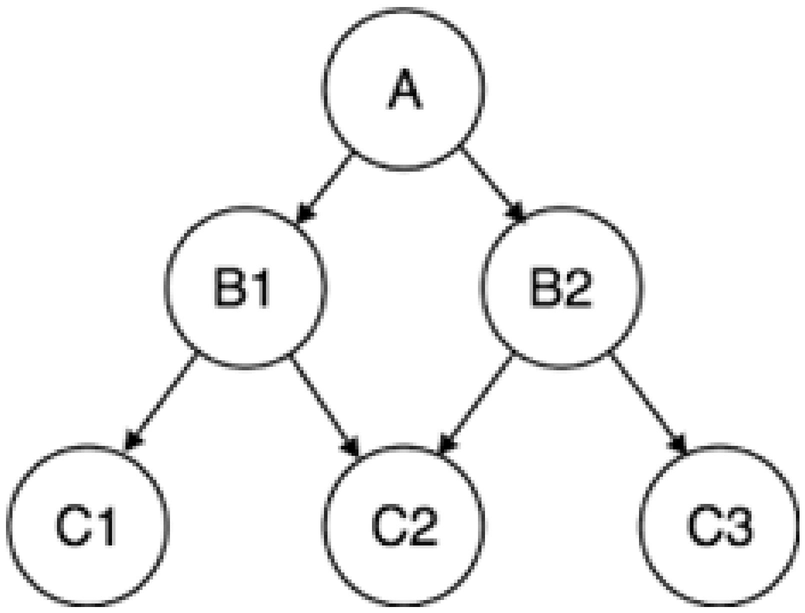
Hierarchical Model-- Hierarchical Database Model, as the name suggests, is a database model in which the data is arranged in a hierarchical tree edifice. As it is arranged based on the hierarchy, every record of data tree should have at least one parent, except for the child records in the last level, and each parent should have one or more child records. The Data can be accessed by following through the classified structure, always initiated from the Root or the first parent. Hence this model is named as Hierarchical Database Model.

What is Hierarchical Database Model

It is a data model in which data is represented in the tree-like structure. In this model, data is stored in the form of records which are the collection of fields. The records are connected through links and the type of record tells which field is contained by the record. Each field can contain only one value.

It must have only one parent for each child node but parent nodes can have more than one child. Multiple parents are not allowed. This is the major difference between the hierarchical and network database model. The first node of the tree is called the root node. When data needs to be retrieved then the whole tree is traversed starting from the root node. This model represents one- to- many relationships.

Let us see one example: Let us assume that we have a main directory which contains other subdirectories. Each subdirectory contains more files and directories. Each directory or file can be in one directory only i.e. it has only one parent.



Here **A** is the main directory i.e. the root node. **B1** and **B2** are their child or subdirectories. B1 and B2 also have two children **C1, C2** and **C2, C3** respectively. They may be directories or other files. This depicts one- to- many relationships.

Uses of Hierarchical Database Model

The uses of the database model are as explained here.

A Hierarchical database model was widely used during the Mainframe Computers Era. Today, it is used mainly for storing file systems and geographic information. It is used in applications where high performance is required such as telecommunications and banking. A hierarchical database is also used for Windows Registry in the Microsoft Windows operating system. It is useful where the following two conditions are met:

1. The data should be in a hierarchical pattern i.e. parent-child relationship must be present.
2. The data in a hierarchical pattern must be accessed through a single path only.

Advantages

Few advantages are listed below.

- Data can be retrieved easily due to the explicit links present between the table structures.

- Referential integrity is always maintained i.e. any changes made in the parent table are automatically updated in a child table.
- Promotes data sharing.
- It is conceptually simple due to the parent-child relationship.
- Database security is enforced.
- Efficient with 1: N relationships.
- A clear chain of command or authority.
- Increases specialization.
- High performance.
- Clear results.

Disadvantages

Below are some of the disadvantages given.

- If the parent table and child table are unrelated then adding a new entry in the child table is difficult because additional entry must be added in the parent table.
- Complex relationships are not supported.
- Redundancy which results in inaccurate information.
- Change in structure leads to change in all application programs.
- M: N relationship is not supported.
- No data manipulation or data definition language.
- Lack of standards.
- Poor flexibility
- Communication barriers
- Organizational Disunity.
- Rigid structure

Features

Some features are pointed out below:

- **Many to many relationships:** It only supports one – to – many relationships. Many to many relationships are not supported.

- **Problem in Deletion:** If a parent is deleted then the child automatically gets deleted.

- **Hierarchy of data:** Data is represented in a hierarchical tree-like structure.

- **Parent-child relationship:** Each child can have only one parent but a parent can have more than one children.

- **Pointer:** Pointers are used for linking records that tell which is a parent and which child record is.

- **Disk input and output is minimized:** Parent and child records are placed or stored close to each other on the storage device which minimizes the hard disk input and output.

- **Fast navigation:** As parent and child are stored close to each other so access time is reduced and navigation becomes faster.

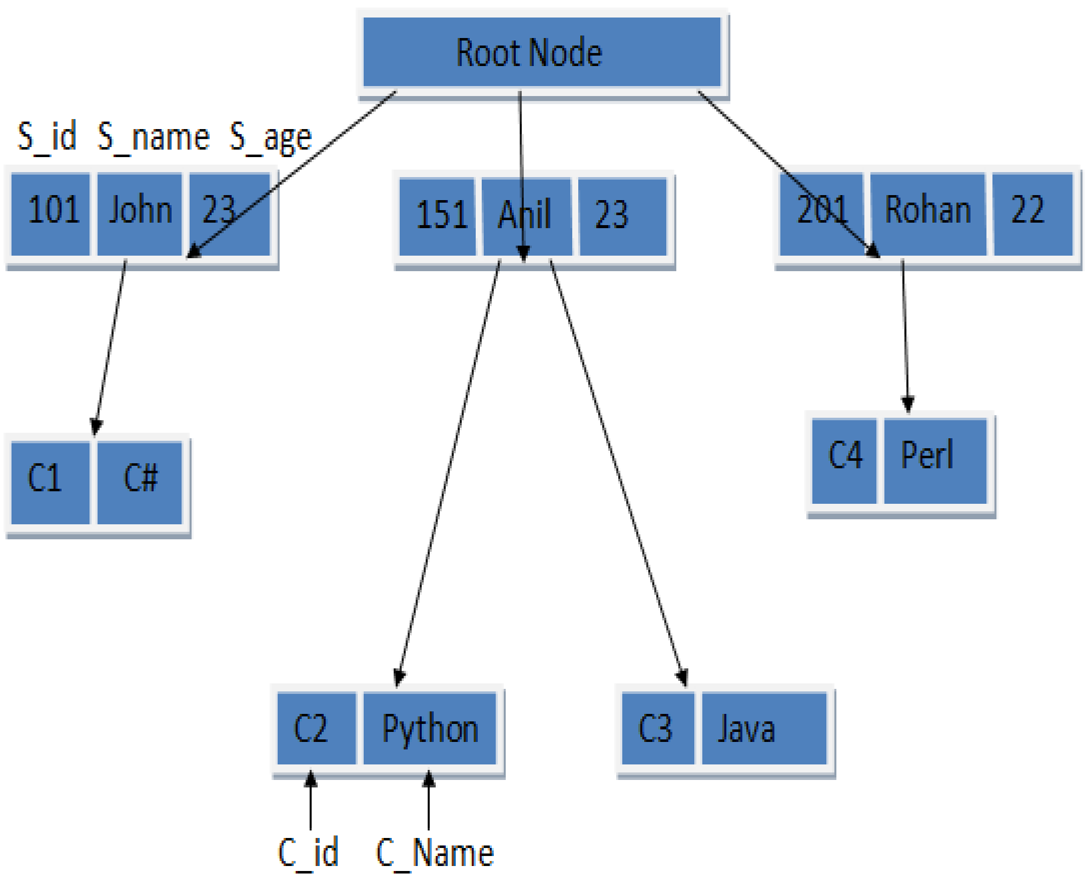
- **Predefined relationship:** All relations between root, parent and child nodes are predefined in the database schema.

- **Re-organization difficulty:** Hierarchy prevents the re-organization of data.

- **Redundancy:** One to many relationships increases redundancy in the data which leads to the retrieval of inaccurate data.

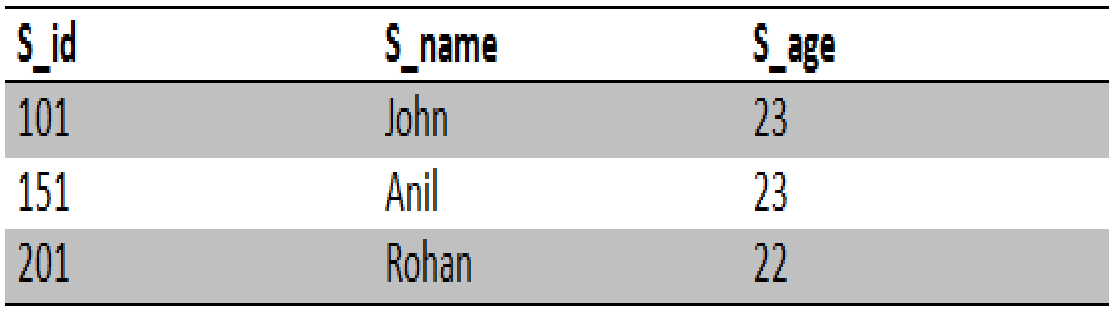
Examples

Let us take an example of college students who take different courses. A course can be assigned to an only single student but a student can take as many courses as they want therefore following one to many



Now we can represent the above hierarchical model as relational tables as shown below:

Student Table:



S_id	S_name	S_age
101	John	23
151	Anil	23
201	Rohan	22

Course Table:

C_id	C_Name	S_id
C1	C#	101
C2	Python	151
C3	Java	151
C4	Perl	201

In this manner, the hierarchical model can be represented in relational tables and vice versa can also be done.

Conclusion

In this article, we have discussed the hierarchical database model in detail which depicts the parent-child relationship which makes it easy to represent data and understand the concept easily. It was mostly used in times of mainframe computers but still, it is used in many fields where high performance and easy concepts are the parameters. So the hierarchical model is efficient for one to many relationships and is widely used in recording file system data.

Network Model

There are many ways to describe and analyze data communications networks. All networks provide the same basic functions to transfer a message from sender to receiver, but each network can use different network hardware and software to provide these functions. All of these hardware and software products have to work together to successfully transfer a message.

One way to accomplish this is to break the entire set of communications functions into a series of layers, each of which can be defined separately. In this way, vendors can develop software and hardware to provide the functions of each layer separately. The software or hardware can work in any manner and can be easily updated and improved, as long as the interface between that layer and the ones around it remain unchanged. Each piece of hardware and software can then work together in the overall network.

There are many different ways in which the network layers can be designed. The two most important network models are the Open Systems Interconnection Reference (OSI) model and the Internet model.

Open Systems Interconnection Reference Model

The Open Systems Interconnection Reference model (usually called the OSI model for short) helped change the face of network computing. Before the OSI model, most commercial networks used by businesses were built using nonstandardized technologies developed by one vendor (remember that the Internet was in use at the time but was not widespread and certainly was not commercial). During the late 1970s, the International Organization for Standardization (ISO) created the Open System Interconnection Subcommittee, whose task was to develop a framework of standards for computer-to-computer communications. In 1984, this effort produced the OSI model.

The OSI model is the most talked about and most referred to network model. If you choose a career in networking, questions about the OSI model will be on the network certification exams offered by Microsoft, Cisco, and other vendors of network hardware and software. However, you will probably never use a network based on the OSI model. Simply put, the OSI model never caught on commercially in North America, although some European networks use it, and some network components developed for use in the United States arguably use parts of it. Most networks today use the Internet model, which is discussed in the next section. However, because there are many similarities between the OSI model and the Internet model, and because most people in networking are expected to know the OSI model, we discuss it here. The OSI model has seven layers (see Figure 1.3).

Layer 1: Physical Layer The physical layer is concerned primarily with transmitting data bits (zeros or ones) over a communication circuit. This layer defines the rules by which ones and zeros are transmitted, such as voltages of electricity, number of bits sent per second, and the physical format of the cables and connectors used.

Layer 2: Data Link Layer The data link layer manages the physical transmission circuit in layer 1 and transforms it into a circuit that is free of transmission errors as far as layers above are concerned. Because layer 1 accepts and transmits only a raw stream of bits without understanding their meaning or structure, the data link layer must create and recognize message boundaries; that is, it must mark where a message starts and where it ends. Another major task of layer 2 is to solve the problems caused by damaged, lost, or duplicate messages so the succeeding layers are shielded from transmission errors. Thus, layer 2 performs error detection and correction. It also decides when a device can transmit so that two computers do not try to transmit at the same time.

OSI Model	**Internet Model**	**Groups of Layers**	**Examples**
7. Application Layer	5. Application Layer	Application Layer	Internet Explorer and Web pages
6. Presentation			

Layer			
5. Session Layer			
4. Transport Layer	4. Transport Layer	Internetwork Layer	TCP/IP Software
3. Network Layer	3. Network Layer		
2. Data Link Layer	2. Data Link Layer	Hardware Layer	Ethernet port, Ethernet cables, and Ethernet software drivers
1. Physical Layer	1. Physical Layer		

Figure 1.3 Network models. OSI = Open Systems Interconnection Reference

Layer 3: Network Layer The network layer performs routing. It determines the next computer the message should be sent to so it can follow the best route through the network and finds the full address for that computer if needed.

Layer 4: Transport Layer The transport layer deals with end-to-end issues, such as procedures for entering and departing from the network. It establishes, maintains, and terminates logical connections for the transfer of data between the original sender and the final destination of the message. It is responsible for breaking a large data transmission into smaller packets (if needed), ensuring that all the packets have been received, eliminating duplicate packets, and performing flow control to ensure that no computer is overwhelmed by the number of messages it receives. Although error control is performed by the data link layer, the transport layer can also perform error checking.

Layer 5: Session Layer The session layer is responsible for managing and structuring all sessions. Session initiation must arrange for all the desired and required services between session participants, such as logging onto circuit equipment, transferring files, using various terminal types, and performing security checks. Session termination provides an orderly way to end the session, as well as a means to abort a session prematurely. It may have some redundancy built in to recover from a broken transport (layer 4) connection in case of failure. The session layer also handles session accounting so the correct party receives the bill.

Layer 6: Presentation Layer The presentation layer formats the data for presentation to the user. Its job is to accommodate different interfaces on different terminals or computers so the application program need not worry about them. It is concerned with displaying, formatting, and editing user inputs and outputs. For example, layer 6 might perform data compression, translation between different data formats, and screen formatting. Any function (except those in layers 1 through 5) that is requested sufficiently often to warrant finding a general solution is placed in the presentation layer, although some of these functions can be performed by separate hardware and software (e.g., encryption).

Layer 7: Application Layer The application layer is the end user's access to the network. The primary purpose is to provide a set of utilities for application programs. Each user program determines the set of messages and any action it might take on receipt of a message. Other network-specific applications at this layer include network monitoring and network management.

Internet Model

Although the OSI model is the most talked about network model, the one that dominates current hardware and software is a more simple five-layer Internet model. Unlike the OSI model that was developed by formal committees, the Internet model evolved from the work of thousands of people who developed pieces of the Internet. The OSI model is a formal standard that is documented in one standard, but the Internet model has never been formally defined; it has to be interpreted from a number of standards.1 The two models have very much in common (see Figure 1.3); simply put, the Internet model collapses the top three OSI layers into one layer. Because it is clear that the Internet has won the "war," we use the five-layer Internet model for the rest of this topic.

Layer 1: The Physical Layer The physical layer in the Internet model, as in the OSI model, is the physical connection between the sender and receiver. Its role is to transfer a series of electrical, radio, or light signals through the circuit. The physical layer includes all the hardware devices (e.g., computers, modems, and hubs) and physical media (e.g., cables and satellites). The physical layer specifies the type of connection and the electrical signals, radio waves, or light pulses that pass through it.

Layer 2: The Data Link Layer The data link layer is responsible for moving a message from one computer to the next computer in the network path from the sender to the receiver. The data link layer in the Internet model performs the same three functions as the data link layer in the OSI model. First, it controls the physical layer by deciding when to transmit messages over the media. Second, it formats the messages by indicating where they start and end. Third, it detects and corrects any errors that have occurred during transmission.

Layer 3: The Network Layer The network layer in the Internet model performs the same functions as the network layer in the OSI model. First, it performs routing, in that it selects the next computer to which the message should be sent. Second, it can find the address of that computer if it doesn't already know it.

Layer 4: The Transport Layer The transport layer in the Internet model is very similar to the transport layer in the OSI model. It performs two functions. First, it is responsible for linking the application layer software to the network and establishing end-to-end connections between the sender and receiver when such connections are needed. Second, it is responsible for breaking long messages into several smaller messages to make them easier to transmit. The transport layer can also detect lost messages and request that they be resent.

Layer 5: Application Layer The application layer is the application software used by the network user and includes much of what the OSI model contains in the application, presentation, and session layers. It is the user's access to the network. By using the application software, the user defines what messages are sent over the network.It discusses the architecture of network applications and several types of network application software and the types of messages they generate.

Groups of Layers The layers in the Internet are often so closely coupled that decisions in one layer impose certain requirements on other layers. The data link layer and the physical layer are closely tied together because the data link layer controls the physical layer in terms of when the physical layer can transmit. Because these two layers are so closely tied together, decisions about the data link layer often drive the decisions about the physical layer. For this reason, some people group the physical and data link layers together and call them the hardware layers. Likewise, the transport and network layers are so closely coupled that sometimes these layers are called the internetwork layer. See Figure 1.3. When you design a network, you often think about the network design in terms of three groups of layers: the hardware layers (physical and data link), the internetwork layers (network and transport), and the application layer.

Message Transmission Using Layers

Each computer in the network has software that operates at each of the layers and performs the functions required by those layers (the physical layer is hardware not software). Each layer in the network uses a formal language, or protocol, that is simply a set of rules that define what the layer will do and that provides a clearly defined set of messages that software at the layer needs to understand. For example, the protocol used for Web applications is HTTP. In general, all messages sent in a network pass through all layers. All layers except the Physical layer add a Protocol Data Unit (PDU) to the message as it passes through them. The PDU contains information that is needed to transmit the message through the network. Some experts use the word "Packet" to mean a PDU. Figure 1.4 shows how a message requesting a Web page would be sent on the Internet.

Application Layer First, the user creates a message at the application layer using a Web browser by clicking on a link (e.g., get the home page at **www.somebody.com**). The browser translates the user's message (the click on the Web link) into HTTP. The rules of HTTP define a specific PDU—called an HTTP packet—that all Web browsers must use when they request a Web page.

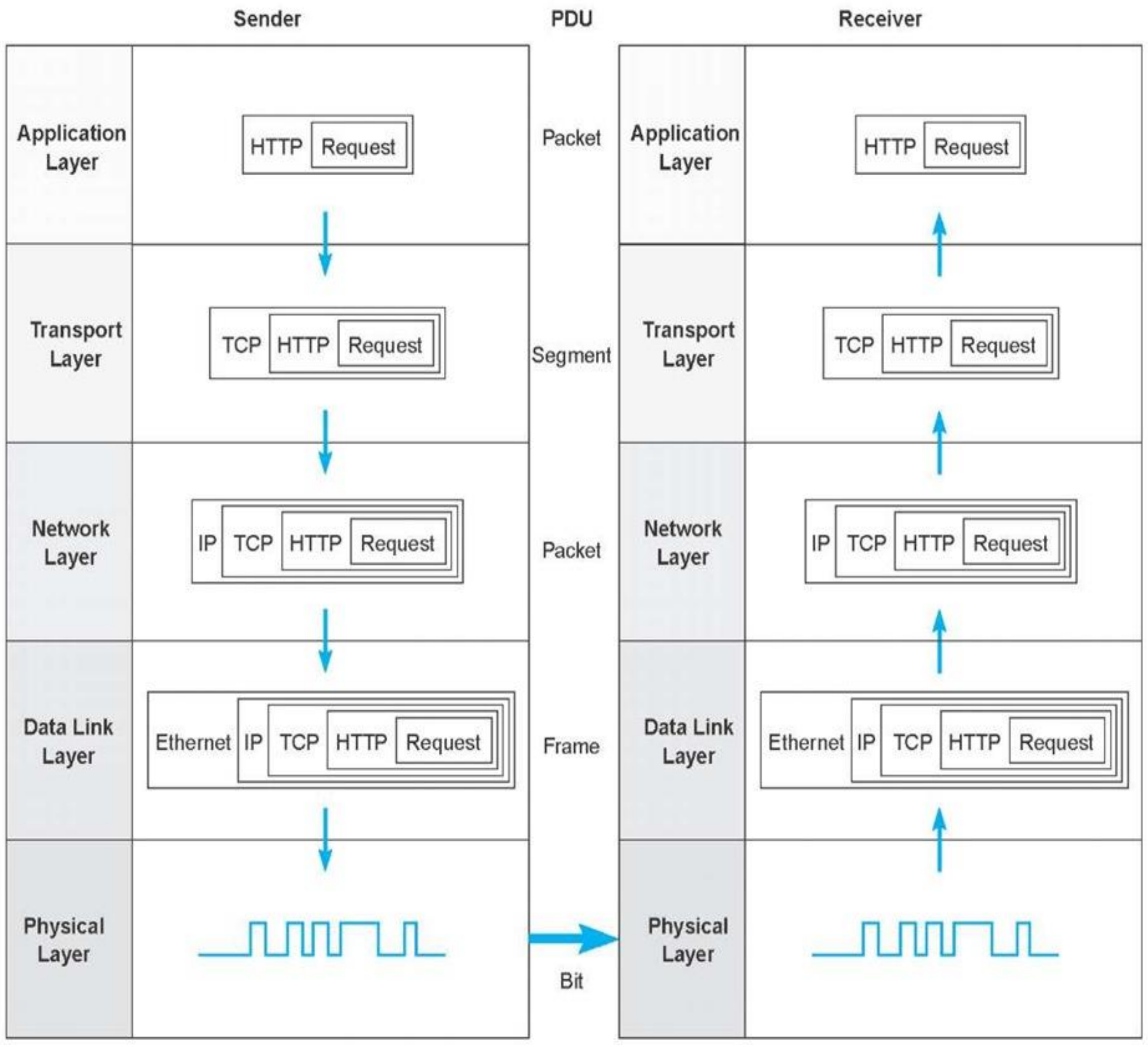


Figure 1.4 Message transmission using layers. IP = Internet Protocol; HTTP/Hypertext Transfer Protocol; TCP = Transmission Control Protocol

For now, you can think of the HTTP packet as an envelope into which the user's message (get the Web page) is placed. In the same way that an envelope placed in the mail needs certain information written in certain places (e.g., return address, destination address), so too does the HTTP packet. The Web browser fills in the necessary information in the HTTP packet, drops the user's request inside the packet, then passes the HTTP packet (containing the Web page request) to the transport layer.

Transport Layer The transport layer on the Internet uses a protocol called TCP (Transmission Control Protocol), and it, too, has its own rules and its own PDUs. TCP is responsible for breaking large files into smaller packets and for opening a connection to the server for the transfer of a large set of packets. The transport layer places the HTTP packet inside a TCP PDU (which is called a TCP segment), fills in the information needed by the TCP segment, and passes the TCP segment (which contains the HTTP packet, which, in turn, contains the message) to the network layer.

Network Layer The network layer on the Internet uses a protocol called IP (Internet Protocol), which has its rules and PDUs. IP selects the next stop on the message's route through the network. It places the TCP segment inside an IP PDU which is called an IP packet, and passes the IP packet, which contains the TCP segment, which, in turn, contains the HTTP packet, which, in turn, contains the message, to the data link layer.

Data Link Layer If you are connecting to the Internet using a LAN, your data link layer may use a protocol called Ethernet, which also has its own rules and PDUs. The data link layer formats the message with start and stop markers, adds error checking information, places the IP packet inside an Ethernet PDU, which is called an Ethernet frame, and instructs the physical hardware to transmit the Ethernet frame, which contains the IP packet, which contains the TCP segment, which contains the HTTP packet, which contains the message.

Physical Layer The physical layer in this case is network cable connecting your computer to the rest of the network. The computer will take the Ethernet frame (complete with the IP packet, the TCP segment, the HTTP packet, and the message) and sends it as a series of electrical pulses through your cable to the server.

When the server gets the message, this process is performed in reverse. The physical hardware translates the electrical pulses into computer data and passes the message to the data link layer. The data link layer uses the start and stop markers in the Ethernet frame to identify the message. The data link layer checks for errors and, if it discovers one, requests that the message be resent. If a message is received without error, the data link layer will strip off the Ethernet frame and pass the IP packet (which contains the TCP segment, the HTTP packet, and the message) to the network layer. The network layer checks the IP address and, if it is destined for this computer, strips off the IP packet and passes the TCP segment, which contains the HTTP packet and the message to the transport layer. The transport layer processes the message, strips off the TCP segment, and passes the HTTP packet to the application layer for processing. The application layer (i.e., the Web server) reads the HTTP packet and the message it contains (the request for the Web page) and processes it by generating an HTTP packet containing the Web page you requested. Then the process starts again as the page is sent back to you.

There are three important points in this example. First, there are many different software packages and many different PDUs that operate at different layers to successfully transfer a message. Networking is in some ways similar to the Russian Matryoshka, nested dolls that fit neatly inside each other. This is called encapsulation, because the PDU at a higher level is placed inside the PDU at a lower level so that the lower level PDU encapsulates the higher level one. The major advantage of using different software and protocols is that it is easy to develop new software, because all one has to do is write software for one level at a time. The developers of Web applications, for example, do not need to write software to perform error checking or routing, because those are performed by the data link and network layers. Developers can simply assume those functions are performed and just focus on the application layer. Likewise, it is simple to change the software at any level (or add new application protocols), as long as the interface between that layer and the ones around it remains unchanged.

Second, it is important to note that for communication to be successful, each layer in one computer must be able to communicate with its matching layer in the other computer. For example, the physical layer connecting the client and server must use the same type of electrical signals to enable each to understand the other (or there must be a device to translate between them). Ensuring that the software used at the different layers is the same is accomplished by using standards. A standard defines a set of rules, called protocols, that explain exactly how hardware and software that conform to the standard are required to operate. Any hardware and software that conform to a standard can communicate with any other hardware and software that conform to the same standard. Without standards, it would be virtually impossible for computers to communicate.

Third, the major disadvantage of using a layered network model is that it is somewhat inefficient. Because there are several layers, each with its own software and PDUs, sending a message involves many software programs (one for each protocol) and many PDUs. The PDUs add to the total amount of data that must be sent (thus increasing the time it takes to transmit), and the different software packages increase the processing power needed in computers. Because the protocols are used at different layers and are stacked on top of one another (take another look at Figure 1.4), the set of software used to understand the different protocols is often called a protocol stack.

Detailed study of Relational Model (Relations, Properties):

Relational Model (RM) represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Some popular Relational Database management systems are:

- DB2 and Informix Dynamic Server - IBM
- Oracle and RDB – Oracle

- SQL Server and Access - Microsoft

- In this tutorial, you will learn Relational Model Concepts
- Relational Integrity Constraints
- Operations in Relational Model
- Best Practices for creating a Relational Model
- Advantages of using Relational Model
- Disadvantages of using Relational Model

Relational Model Concepts

1. **Attribute:** Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME,etc.

2. **Tables** – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.

3. **Tuple** – It is nothing but a single row of a table, which contains a single record.

4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.

5. **Degree:** The total number of attributes which in the relation is called the degree of the relation.

6. **Cardinality:** Total number of rows present in the Table.

7. **Column:** The column represents the set of values for a specific attribute.

8. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.

9. **Relation key** - Every row has one, two or multiple attributes, which is called relation key.

10. **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain

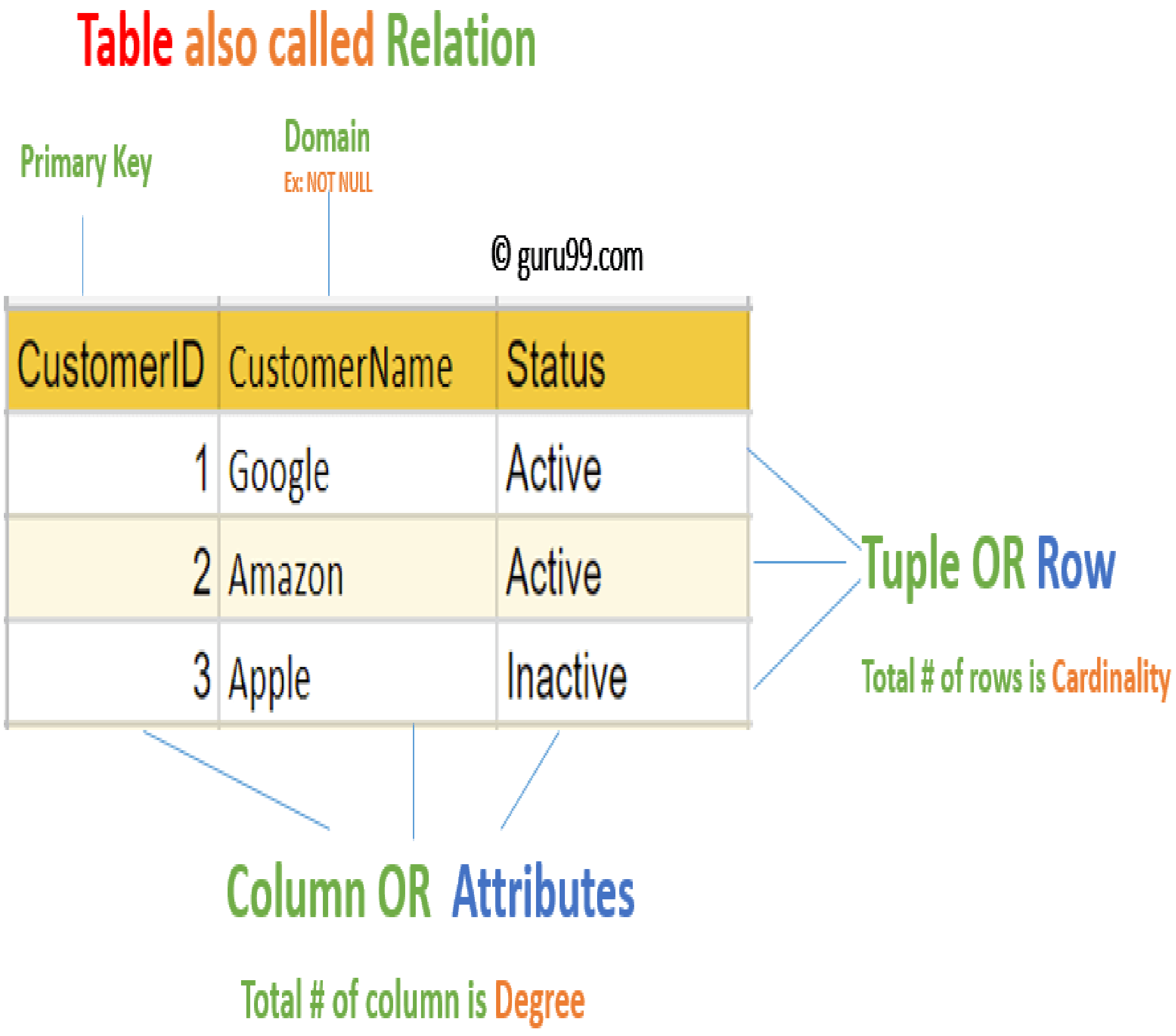


Table also called Relation
Primary Key
Domain
Ex: NOT NULL
© guru99.com
Tuple OR Row
Total # of rows is Cardinality
Column OR Attributes
Total # of column is Degree

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Relational Integrity Constraints

Relational Integrity constraints in DBMS are referred to conditions which must be present for a valid relation. These Relational constraints in DBMS are derived from the rules in the mini-world that the database represents.

There are many types of Integrity Constraints in DBMS. Constraints on the Relational database management system is mostly divided into three main categories are:

1. Domain Constraints
2. Key Constraints
3. Referential Integrity Constraints

Domain Constraints

Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type.

Domain constraints specify that within each tuple, and the value of each attribute must be unique. This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc.

Example:

```
Create DOMAIN CustomerName
CHECK (value not NULL)
```

The example shown demonstrates creating a domain constraint such that CustomerName is not NULL

Key Constraints

An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

Example:

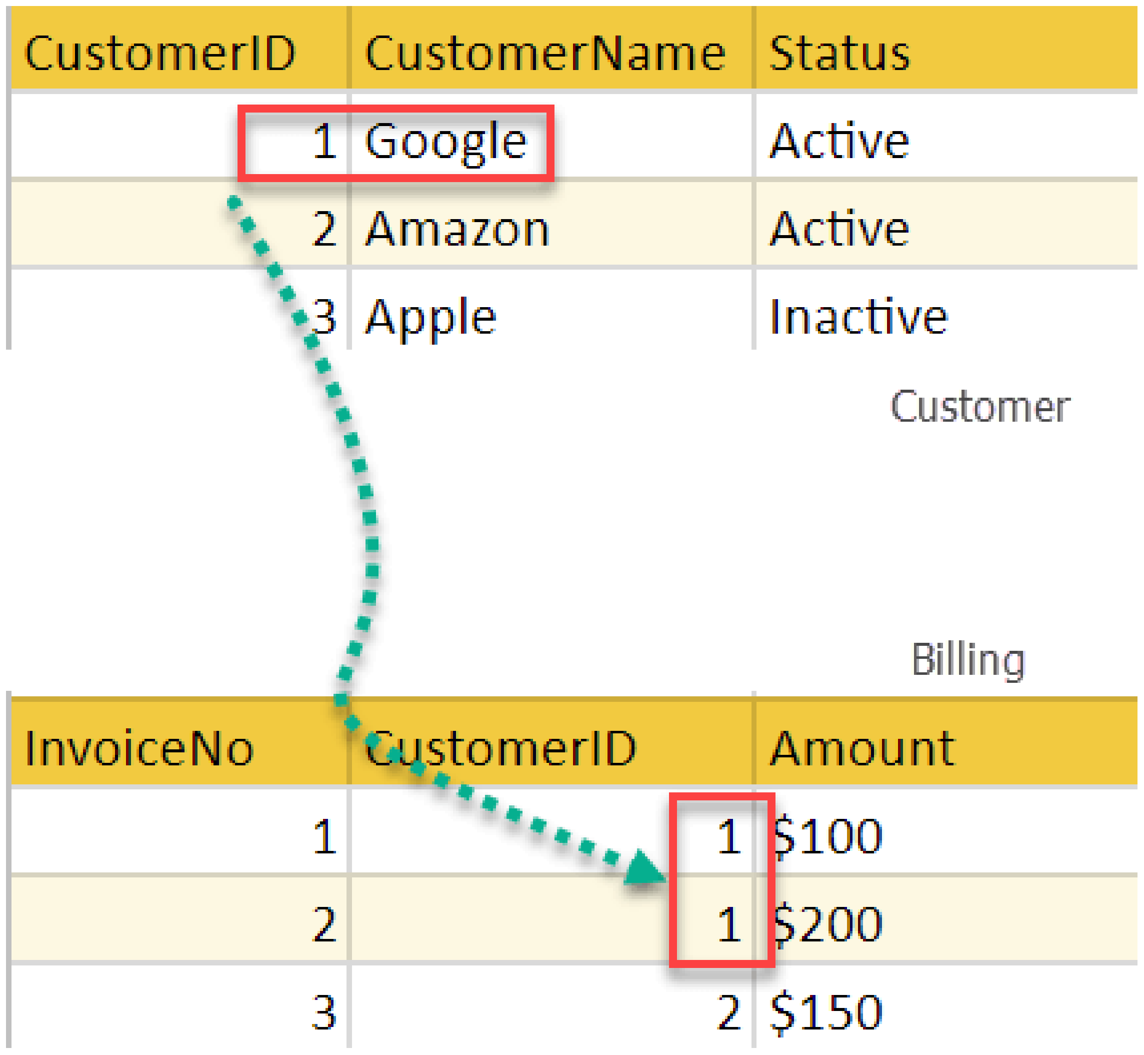
In the given table, CustomerID is a key attribute of Customer Table. It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName =" Google".

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Referential Integrity Constraints

Referential Integrity constraints in DBMS are based on the concept of Foreign Keys. A foreign key is an important attribute of a relation which should be referred to in other relationships. Referential integrity constraint state happens where relation refers to a key attribute of a different or same relation. However, that key element must exist in the table.

Example:



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Customer

Billing

InvoiceNo	CustomerID	Amount
1	1	$100
2	1	$200
3	2	$150

In the above example, we have 2 relations, Customer and Billing.

Tuple for CustomerID =1 is referenced twice in the relation Billing. So we know CustomerName=Google has billing amount $300

Operations in Relational Model

Four basic update operations performed on relational database model are

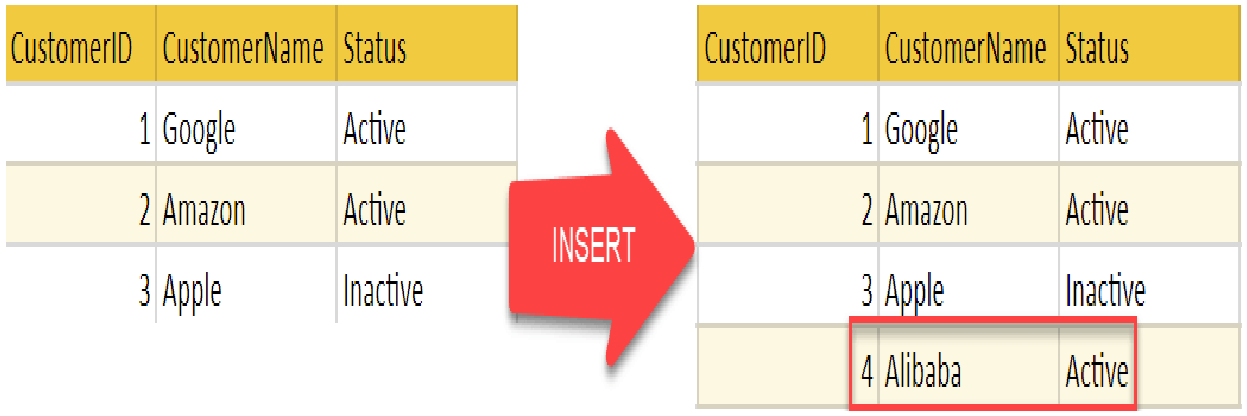
Insert, update, delete and select.

- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Modify allows you to change the values of some attributes in existing tuples.
- Select allows you to choose a specific range of data.

Whenever one of these operations are applied, integrity constraints specified on the relational database schema must never be violated.

Insert Operation

The insert operation gives values of the attribute for a new tuple which should be inserted into a relation.

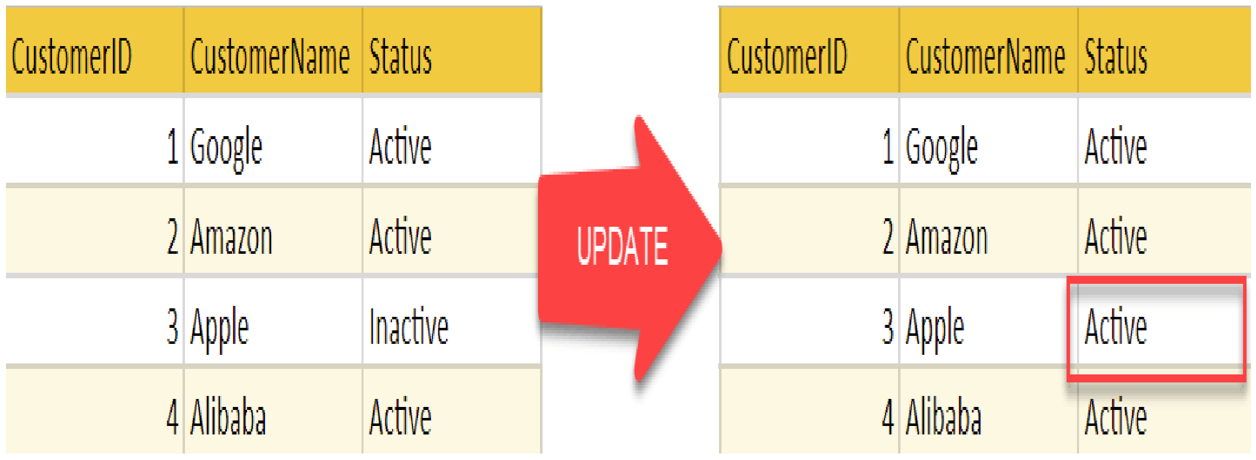


CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Update Operation

You can see that in the below-given relation table CustomerName= 'Apple' is updated from Inactive to Active.

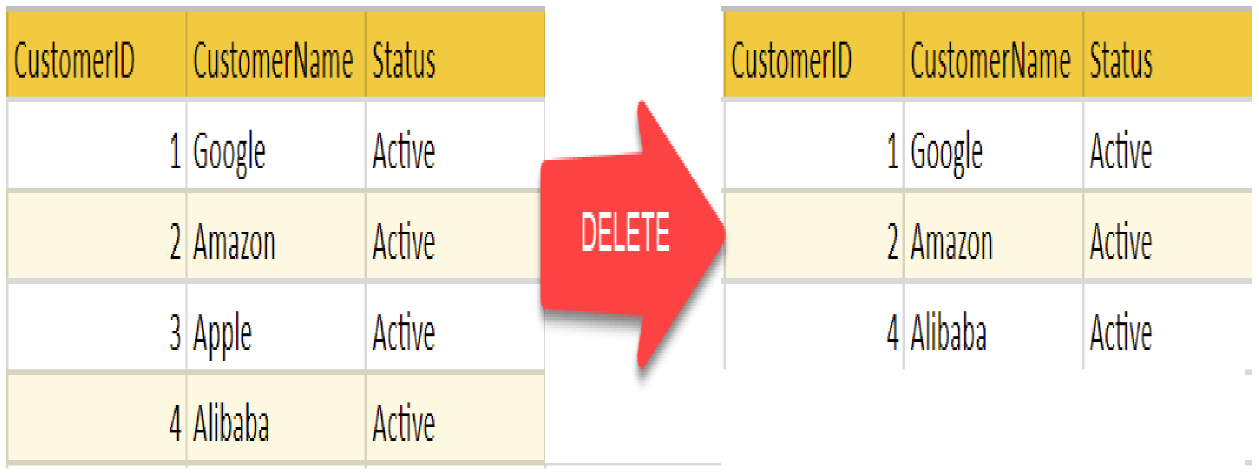


CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active

Delete Operation

To specify deletion, a condition on the attributes of the relation selects the tuple to be deleted.



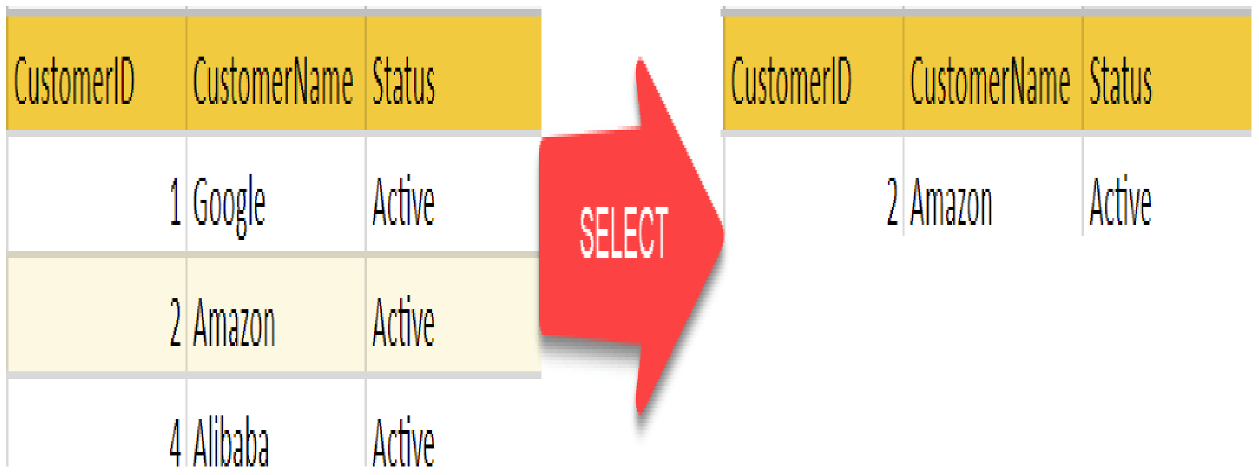
CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active

In the above-given example, CustomerName= "Apple" is deleted from the table.

The Delete operation could violate referential integrity if the tuple which is deleted is referenced by foreign keys from other tuples in the same database.

Select Operation



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active

CustomerID	CustomerName	Status
2	Amazon	Active

In the above-given example, CustomerName="Amazon" is selected

Best Practices for creating a Relational Model

- Data need to be represented as a collection of relations

- Each relation should be depicted clearly in the table
- Rows should contain data about instances of an entity
- Columns must contain data about attributes of the entity
- Cells of the table should hold a single value
- Each column should be given a unique name
- No two rows can be identical
- The values of an attribute should be from the same domain

Advantages of using Relational Model

- **Simplicity**: A Relational data model in DBMS is simpler than the hierarchical and network model.
- **Structural Independence**: The relational database is only concerned with data and not with a structure. This can improve the performance of the model.
- **Easy to use**: The Relational model in DBMS is easy as tables consisting of rows and columns are quite natural and simple to understand
- **Query capability**: It makes possible for a high-level query language like SQL to avoid complex database navigation.
- **Data independence**: The Structure of Relational database can be changed without having to change any application.
- **Scalable**: Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

Disadvantages of using Relational Model

- Few relational databases have limits on field lengths which can't be exceeded.
- Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.
- Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another.

Summary

- The Relational database modelling represents the database as a collection of relations (tables)
- Attribute, Tables, Tuple, Relation Schema, Degree, Cardinality, Column, Relation instance, are some important components of Relational Model
- Relational Integrity constraints are referred to conditions which must be present for a valid Relation approach in DBMS
- Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type
- Insert, Select, Modify and Delete are the operations performed in Relational Model constraints
- The relational database is only concerned with data and not with a structure which can improve the performance of the model
- Advantages of Relational model in DBMS are simplicity, structural independence, ease of use, query capability, data independence, scalability, etc.
- Few relational databases have limits on field lengths which can't be exceeded.

Properties Model

Property is a fundamental national asset, the use of which has an important influence on the environment, the economy, and society in general. In recognition of this, the Department of Construction Economics and Management has responded to the demand from both students and employers for postgraduate courses in property and real estate studies with the introduction of taught postgraduate courses in Property Studies. The programmes consist of taught modules, project work and, in the case of the Masters programme, a research component.

The MSc programme is fully accredited by both the South African Council* for the Property Valuers Profession and the Royal Institution of Chartered Surveyors.

* requires both CON5043Z (Property Valuation Theory and Practice) and CON5044Z (Advanced Property Valuation) for accreditation purposes.

Structure of the Programme

The programme is presented over three years and consists of two self-study courses and six compulsory taught modules, followed by the Research Report. Each taught module consists of lectures, group discussions and assignments, reading and interaction with experts in specific fields.

Course Modules

The six compulsory modules are presented every year at approximately twelve week intervals, commencing in January/February. Exact dates may be obtained from the department. The MSc in Property Studies runs over three years. Each module typically comprises 20 hours of pre-reading, a 40-hour contact week at the university and 120 hours of assignment work. Candidates registered for the MSc in Property Studies is required to take the six compulsory modules.

For the 20 credit courses, there is no set order in which these must be taken. However, the first courses that a student must register for are CON5036Z (Introduction to Research) and CON5041Z (Introduction to Applied Statistics). These short courses are delivered through directed learning of set materials and are concerned with ensuring that every student on the MSc course is familiar with writing in an academic style and has a familiarity of interpreting basic statistics. Two short exams are then held at the beginning of the first 20 credit module in January and again in June. Passing both these short courses is a requirement for completing the first 20 credit module because these courses are designed to support the student in the remaining courses. CON5037Z (Research Methodology) is also a pre-requisite to completion of CON5023Z (Research Report) as the student will use the knowledge gained in CON5037Z in his/her research.

The compulsory modules are as follows:

- **Property Development – CON5006Z (20 credits)**

  Investment Evaluation. Environmental Impact Assessment. Risk Assessment. Land Assembly and Servicing. Economic Viability Analysis. The Construction Stage. Marketing of Improvements. Whole Life Appraisal.

- **Property Law – CON5007Z (20 credits)**

  The Meaning and Function of Law and Legal Rules; the Main Divisions of the Law; the Structure of the Courts, Officers of the Courts and Different Court Procedures; Sources of South African Law; Basic Concepts of Private Law; an Outline of South Africa’s Constitution; the Bill of Rights and Land Use; the Expropriation Act; the Impact of the Environmental Clause and Environmental Legislation on Land Use; Sectional Title & Share Block Schemes; General Principles of the Law of Contract; Specific or Applied Contracts; Sale and Lease; Forms of Security: Contractual and Property Rights; Insolvency Law: The Effect of Insolvency on Property and Uncompleted Contracts; Commercial Agency: Estate Agents; Alternative Dispute Resolution; Case Studies.

- **Urban Land Economics – CON5008Z (20 credits)**

  Urban economics and urban problems. The urbanisation process. The urban hierarchy. Urban rent. Theories of urban spatial structure. Location theory. Problems in developing countries. Time value of money. Discounted Cashflows. Nature and scope of valuation; concepts and theory of value; determinants of value; the valuation process and methods of valuation; law relating to rating; expropriation and property valuation.

- **Property Finance – CON5009Z (20 credits)**

  Mathematics of finance; Property Taxation; Overview of Managerial Finance and Theory; Working Capital management; Long-term Asset Management; Property Investment Decision-making; The Financing Decision and Capital Structure; Capital markets; Sources and Flows of Capital for Property investments; Types of Financial Instruments; Case Studies. Investment Evaluation; Environmental Impact Assessments; Risk Assessment; Land Assembly and Servicing; Economic Viability Analysis; the Construction Stage; Marketing of Improvements; Whole Life Appraisal.

- **Property Valuation Theory & Practice – CON5043Z**

  Introduction: The Valuer; Valuation Theory – concepts and historical development; Accuracy of Valuations; The Surveyor General; Register of Deeds; Local Authorities; Town Planning Schemes; the Valuer’s Records; Factors Affecting Supply and Demand in the Property Market; Different Types of Fixed Property; Factors Influencing the Value of Property; Approaches to the Valuation of Property; the Valuation Report. Potential and its Influence on Value: Legal Concept of Potential; Economic Concept of Potential; Potential for an Alternative Use; Redevelopment Potential; Quantifying the Influence of Potential on Value; Highest and Best Use of a Property; Under-improved Property; Over-improved Property; “Wrong” or Inappropriate Development; Influence of Re-zoning on Value. Methods of Valuation I: Sales, Cost and Income Methods of Valuation. Valuation of Residential Properties: Definition of a Residential Property; Valuation Approach; Sources of Information; the Valuation Process; Limitations on Use and Development; Unimproved Properties; Improved Properties; Valuation of Township Developments including Developers’ Interests. Valuation of Income Producing Properties I: Influence of Leases on Value; Valuation of Leasehold Interests; Valuation of Income Producing Properties; Overview of Capitalisation Rates and their Use in the Valuation of Income Producing Properties. South African Legislative Environment: Relevant legislation and its application to the Valuation Process. Case Law: Relevant Case Law as it pertains to the Valuation of Property. Expropriation: Legislation; Valuation for

Expropriation; Valuation of Servitudes. ARGUS – Valuation DCF Software: Use of the ARGUS software for the valuation of property.

- **Property Portfolio Management – CON5021Z (20 credits)**

  Portfolio management: The Property Cycle; The Economic Cycle; Modern Portfolio Theory; The Property Portfolio. Operational Property/Asset Management: Introduction to Property Management; Legal Aspects/Tenant Issues; Maintenance/Services; Investment Strategy and Value; Current trends; Case studies. Strategic Property/Asset Management; Shopping Centre Management: Management; Leasing; Marketing; Financial Control. Facilities Management: Space planning and management; Relocation; Maintenance management & Life cycle costing. Energy management; Environmental issues. Outsourcing.

- **Minor Dissertation Property Studies – CON5010Z (60 credits)**

  Statistics: data modelling using the Statistica software package. Methodology: selection of the research problem; preparation of the research proposal. Research Report: conducting empirical research; analysis of findings; drawing conclusions; making recommendations; presentation of a research report.

- **Introduction to Research – CON5036Z (4 credits)**

  Research and writing skills; plagiarism; research ethics; critical analysis of literature; creating an argument; writing in an academic style; referencing conventions.

- **Research Methodology – CON5037Z (6 credits)**

  Research methodology, the research experience; knowledge and problems; the proposal chapter; designing the research; theoretical frameworks; overview of research methods – from quantitative to qualitative; case studies; writing the literature review, data presentation and analysis; concluding the research.

- **Introduction to Applied Statistics – CON5041Z (4 credits)**

  Data presentation: Identifying an appropriate population; drawing a sample from the population; organising data; discreet and continuous data types; graphical presentation of data. Descriptive statistics: exploratory data analysis and summary statistics. Applied mathematics: simple interest; equivalence; compound interest; present value; annuities; general annuities; sinking funds; amortization.

- **Further Applied Statistics – CON5042Z (6 credits)**

  Design of a questionnaire: defining the “target” population, drawing a sample from the population, organising the data into an appropriate format for further analysis. Presenting the results: Summarizing the data, and interpreting the results. Statistical methods: Contingency tables; Chi Square tests; multiple regression; t-test and Anova; confidence interval equivalence.

- **Additional Core Courses:**

  To achieve registration with the South African Council for Property Valuers Professions, a student, in addition to the core courses, will have to complete the following one elective:

- **Advanced Property Valuation – CON5044Z (20 credits)**

  Valuation of Income Producing Properties II: Valuation of Residential, Commercial and Industrial Properties; Capitalisation Rates – Detailed Discussion of Capitalisation Rates; Usage and Derivation from Market; Pitfalls. Methods of Valuation II: Residual and Accounts Methods of Valuation. Valuation of Special Properties: Valuation of Sectional Titles; Valuation for Fractional Ownership; Valuation of Farms and Agricultural Land; Valuation of Shopping Centres; Valuation of Special Properties, including Petrol Stations, Air Space, Mining Rights and Minerals, Industrial Plant and Machinery; Non-Negotiable Properties, and Properties Subject to Particular Legislation. Introduction to Non-Market Valuation Methods: Travel Cost Method; Contingent Valuation Method; and Hedonic Pricing Method. Valuations for Rating Purposes (Municipal / Mass Valuations): Fiscal Requirements; Legislative Framework; Valuation Process; Appeals Process. South African Legislative Environment: Relevant Legislation and its application to the Valuation Process. Case Law: Relevant Case Law as it pertains to the Valuation of Property. Issues in Valuation Theory and Practice: Contemporary and Emerging Issues in Valuation Theory and Practice; Developing World Issues. Valuation for Insurance Purposes: Types of Property Insurance; Purpose of Insurance; Insurance Cover; Methods of Estimating and Sources of Cost Data; Inclusions in a Cost Estimate; Location; Professional Fees; Demolition Costs and Site Clearance. GIS: Type of GIS systems; Application of GIS systems to property. Valuation and Listed Property: Understand the relationship between property valuation and listed property.

Entry Requirements

Four-year bachelor or honours degree in an appropriate field, obtained from a recognised University. Examples of appropriate qualifications include, but are not limited to: construction management; quantity surveying; architecture; engineering; and planning. In addition, candidates should have work experience and should preferably be currently employed.

Applicants who do not meet this requirement, may be accepted if they possess recognised tertiary qualifications and have appropriate experience.

Key & Integrity rules

- Integrity rules are needed to inform the DBMS about certain constraints in the real world.
- Specific integrity rules apply to one specific database.
  Example: part weights must be greater than zero.
- General integrity rules apply to all databases.
  Two general rules will be discussed to deal with: primary keys and foreign keys.

PRIMARY KEYS

- Primary key is a unique identifier for a relation.
- There could be several candidate keys, as long as the they satisfy two properties:

  1. uniqueness
  2. minimality

- From the set of candidate keys, one is chosen to be the primary key.
- The others become alternate keys.

EXAMPLE:

The relation R has several candidate keys.
ID SSN License_Number NAME
If we select ID to be the primary key, then the other candidate keys become alternate keys.

THE ENTITY INTEGRITY RULE

- No component of the primary key of a base relation is allowed to accept nulls.

WHAT ARE NULLS?

Null may mean "property does not apply". For example, the supplier may be a country, in which case the attribute CITY has a null value because such property does not apply. Null may mean "value is unknown". For example, if the supplier is a person, then a null value for CITY attribute means we do not know the location of this supplier.

Nulls cannot be in primary keys, but can be in alternate keys.

EXAMPLE:

SSN may be null for one and only one person (why?)

FOREIGN KEYS

A foreign key is an attribute of one relation R2, whose values are required to match those of the primary key of some other relation R1 (R1 and R2 can be identical)

EXAMPLE:

SP relation has attribute S#, and S relation has primary key S#. Then S# in SP is considered a foreign key.

SP is called the "referencing relation". S is called the "referenced relation".

We can draw a "referential diagram"

SP ---S#---> S

or simply

SP --------> S

WHY ARE FOREIGN KEYS IMPORTANT?

Foreign-to-primary-key matching are the "glue" which holds the database together. Another way of saying it

Foreign keys provide the "links" between two relations.
A relation's foreign key can refer to the same relation.
EXAMPLE: EMP (EMP#, SALARY, MGR_EMP#, ...)
EMP# is the primary key MGR_EMP# is the foreign key
EMP is a "self-referencing relation".

THE REFERENTIAL INTEGRITY RULE
The database must not contain any unmatched foreign key values.

REFERENTIAL INTEGRITY RULE

EXAMPLE:

The three 3NF relations are:
SP(S#,P#,QTY)
SC(S#,CITY)
CS(CITY,STATUS)

The referential diagrams are:

SP ---S#---> SC ---CITY---> CS
DELETE INTEGRITY RULE:
We should not delete (S5,London) from SC if S5 is present in SP.
INSERT INTEGRITY RULE:
We should not insert (S3,P2,200) into SP unless S3 is present in SC.

Comparison of Hierarchical

Was developed in the 1960s. The Hierarchical model was essentially born from the first mainframe database management system. It uses an upside-down tree to structure data. The top of the tree is the parent and the branches are children. Each child can only have one parent but a parent can have many children.

Advantages

- Have many different structures and forms.
- Structures data in an upside-down tree. (Simplifies data overview)

- Manages large amounts of data.
- Express the relationships between information.
- Many children per parent.
- Distribute data in terms of relationships.
- Improve data sharing.

Disadvantages

- One parent per child.
- Complex (users require physical representation of database)
- Navigation system is complex.
- Data must be organized in a hierarchical way without compromising the information.
- Lack structural independence.
- Many too many relationships not supported.
- Data independence.

NETWORK DATA MODEL

In 1965 C.W. Bachman developed the first network data model to present complex data relationships more effectively than the hierarchical model. He tried to impose a database standard with his model and also wanted to improve database performance.

It was in 1971 that the Conference on Data System Languages or CODASYL officially or formally defined the Network model. The network databases arrange its data as a directed graph and have a standard navigational language.

Advantages

- Multi-parent support.
- Somewhat same simplicity as the hierarchical model.
- More useful than the hierarchical data model.

- Deals with even larger amounts of information than the hierarchical model.
- Promotes data integrity.
- Many too many relationships support.
- Data independence.
- Improved data access.

Disadvantages

- Data relationships must be predefined.
- Much more complex than the hierarchical date model.
- Users are still require to know the physical representation of the database
- Information can be related in various and complicated ways.
- Lack structural independence.

RELATIONAL DATA MODEL

The relational data model was introduced in 1970 by Edgar F. Codd. He worked for IBM. All data is represented as simple tabular data structures which the user can access through a high-level non-procedural language. In 1974 IBM proposed a new high-level non-procedural language – SEQUEL (renamed into SQL in 1990).

Advantages

- Structured independence is promoted.
- Users do not have to know the physical representation of the database.
- Use of SQL language to access data.
- Easier database design.
- Tabular view improves simplicity.
- Support large amounts of data.
- Data independence.

- Multi-level relationships between data sets
- No need to predefined data relationships.

Disadvantages

- Data anomalies.
- People need training if they want to use the system effectively and efficiently.

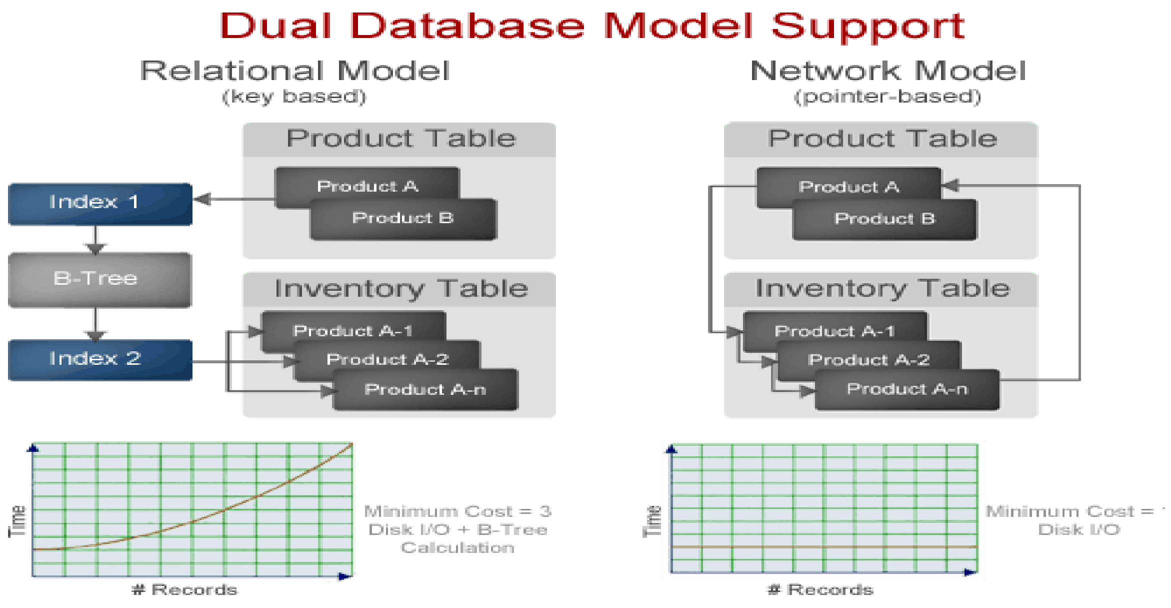
ENTITY RELATIONSHIP DATA MODEL

Dr. Peter Pin-Shan Chen introduced the entity relationship data model in 1976. It is a graphical representation of entities that became popular very quickly because it complemented the relational database model concepts.

Network and Relational Model

We analyze the pros and cons of the relational and network database model. How each model works and highlights the strengths, weaknesses, and capabilities within each model. The differences between these two models can lead to the success or failure in developing an application.

The following discusses how each model works and highlights the strengths, weaknesses, and capabilities within each model. The differences between these two models can lead to the success or failure in developing an application.



Relational Database Model

You are sitting on the bus, headed home. A little tired but not all that sleepy, you decide to listen to some Bon Jovi. On second thought, you want to watch one of his movies (Cry Wolf comes to mind). The iPhone’s spotlight search is nice. Type in “Bon Jovi” as the keyword, and the media player will seamlessly find both their music and movies – two different categories – stored on the device.

Common operations like the ones above usually make use of a relational database management system (RDBMS). The database, let’s call it Media Collection, defines three tables ARTIST, ALBUM and MOVIE where the names of the artists, albums and movie titles are stored respectively. When a particular artist is selected, like Bon Jovi in our example, a SQL query is issued to retrieve all the albums and movie titles that belong to the selected artist. The data returned is displayed on the iPhone screen, usually in alphabetical order.

Relational Data Model Weaknesses

These seemingly simple steps reveal two fundamental weaknesses inherent with the relational data model. The first weakness is the fact that each relationship requires duplicate columns in both tables associated with it. For example, both the ARTIST and ALBUM tables must contain and thus maintain a column that stores the names of the artists so a link between an artist and their albums can be established. This means extra storage space as well as programming overhead are required to keep the two columns "in sync." Changing the name of an artist means that all the ALBUM entries for that artist need to have their "artist name" column updated.

The second weakness, and the more relevant aspect of this article, is the fact that a single relationship may only contain two tables, the primary key (main) table and the foreign key (referencing) table. Within the Music Collection database, both the ALBUM table and the MOVIE table need to reference the ARTIST table. Due to this limitation, two separate relationships need to be defined; one that includes the ALBUM table reference to the ARTIST table and one that has the MOVIE table referencing the ARTIST table.

Inefficient Data Retrieval Operation - Relational Data Model

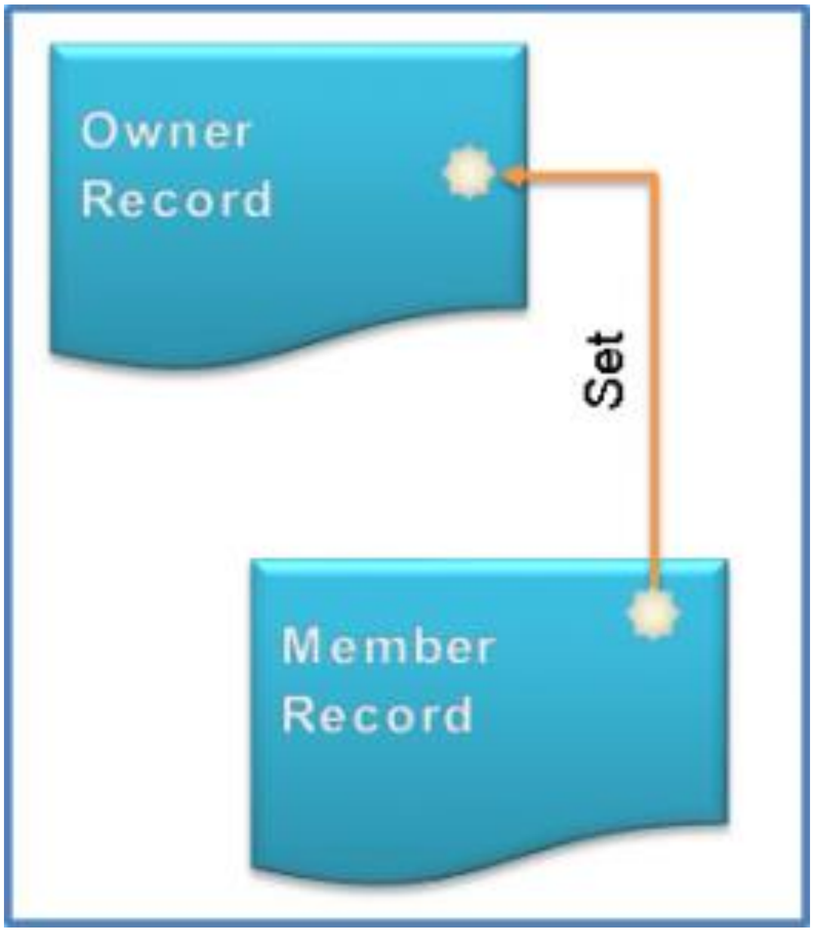
This translates into a rather inefficient data retrieval operation when finding all the albums and movies associated with an artist. During the first operation, the database system retrieves all the related albums from the ALBUM table and stores the result set in a temporary location. During the second operation, the same process as the first is performed, only this time it retrieves results from MOVIES. The final operation merges the two result sets, re-orders them if necessary, and then returns the merged result set.

The inefficiency of the relational model may not be a showstopper when the amount of data in the database is relatively small and there is abundant computing resource available, especially since it is not uncommon today for an average person to own a computer with a 2GHz dual-core CPU with 2GB of memory and a 500GB hard disk. On the other hand, there exists a rapidly emerging market for small computers that require a database system (DBMS) – smartphones, portable music players, and GPS devices, to name a few. With limited CPU speed and memory, these hand-held systems can benefit greatly from a DBMS that allows for flexible and efficient data storage and retrieval in terms of both performance and disk usage.

The network database model offers exactly that.

Network Database Model

An enhanced form of the hierarchical data model, the network model represents data in a tree of records. Relationships between tables (records) are expressed as sets. A set has one parent record (owner) and one or more child records (members). Related records in a set are directly linked by pointers, rather than by matching duplicate columns as is the case in the relational data model.



Records Associated with a Single Owner

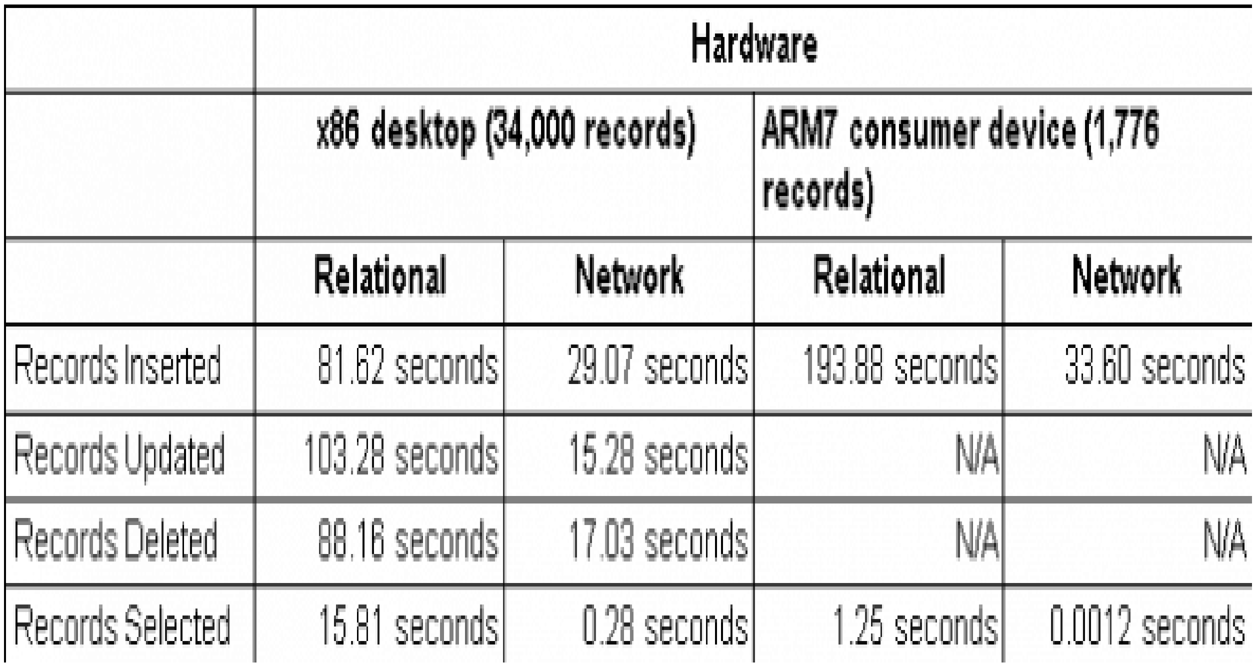
The network database model allows records from more than one table to be associated with a single owner record of another table. This provides a definite advantage over the relational counterpart when querying results from multiple foreign-key tables associated with one primary-key table. In the Media Collection database, both the ALBUM and

MOVIE records can also be members of the ARTIST record in one set, as shown in Figure 2. This means that both albums and movies for a given artist can be retrieved in a single operation. This eliminates the need to store and potentially re-order temporary results in the middle of the operation, resulting in better query performance. Without the need to store and maintain duplicate columns network databases also help reduce disk space and memory usage.

Performance Case Study

Real-life data shows that the performance gain and resource savings of using network databases can be quite significant. In a data structure using a three-way relationship among the ARTIST, ALBUM and SONG tables, our SQL developers compared the data modification and query performance of the relational and network database models using both desktop systems and small, consumer devices. They found that the network model used 29 percent less disk space to store the same number of records and relationships than the relational data model. All the storage savings can be attributed to replacing the ARTIST-ALBUM and the ALBUM-SONG foreign-key indices with set pointers.

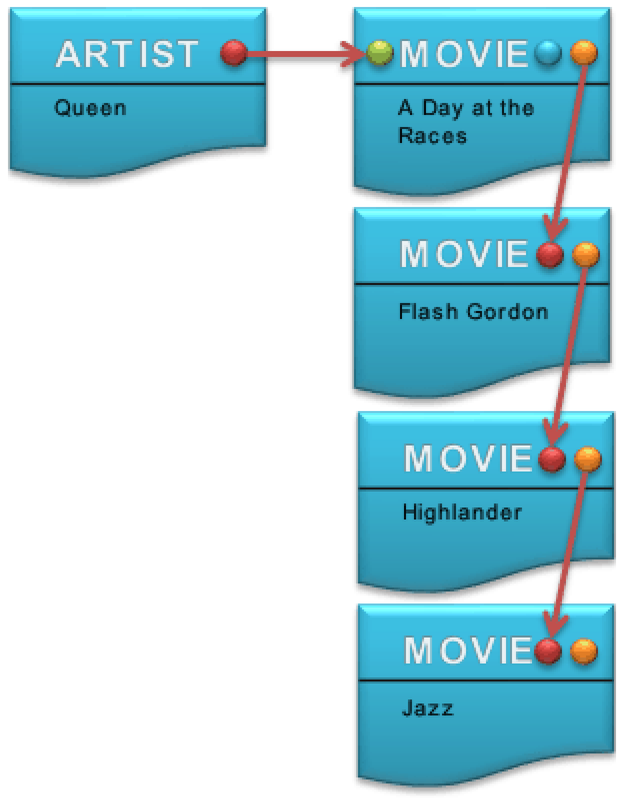
Removing these data structures had a huge effect on the storage requirements because a typical B-Tree index requires approximately 1.3 times the space of what it indexes. They also discovered that the network database model achieved up to 23 times better insert performance and up to 123 times faster query performance, as shown in Table 1.



	Hardware			
	x86 desktop (34,000 records)		ARM7 consumer device (1,776 records)	
	Relational	Network	Relational	Network
Records Inserted	81.62 seconds	29.07 seconds	193.88 seconds	33.60 seconds
Records Updated	103.28 seconds	15.28 seconds	N/A	N/A
Records Deleted	88.16 seconds	17.03 seconds	N/A	N/A
Records Selected	15.81 seconds	0.28 seconds	1.25 seconds	0.0012 seconds

Table 1 – Benchmark results from relational and network models on x86 and ARM7 systems.

Different management requirements mean different data structures and different methods of storing and accessing the data. The resulting system may consist of a few tables with no relationships, or hundreds of tables associated with complex relationships. While the relational data model is the de facto standard, we now know that it does not always provide the optimal solutions for more complex data management problems. Selecting the appropriate data model, or even combining multiple models, can produce a far more efficient result than the relational data model alone. The result is significant cost savings, improving quality and an enhanced user experience.



Conclusion - Network Model for Speed, Relational for Usability

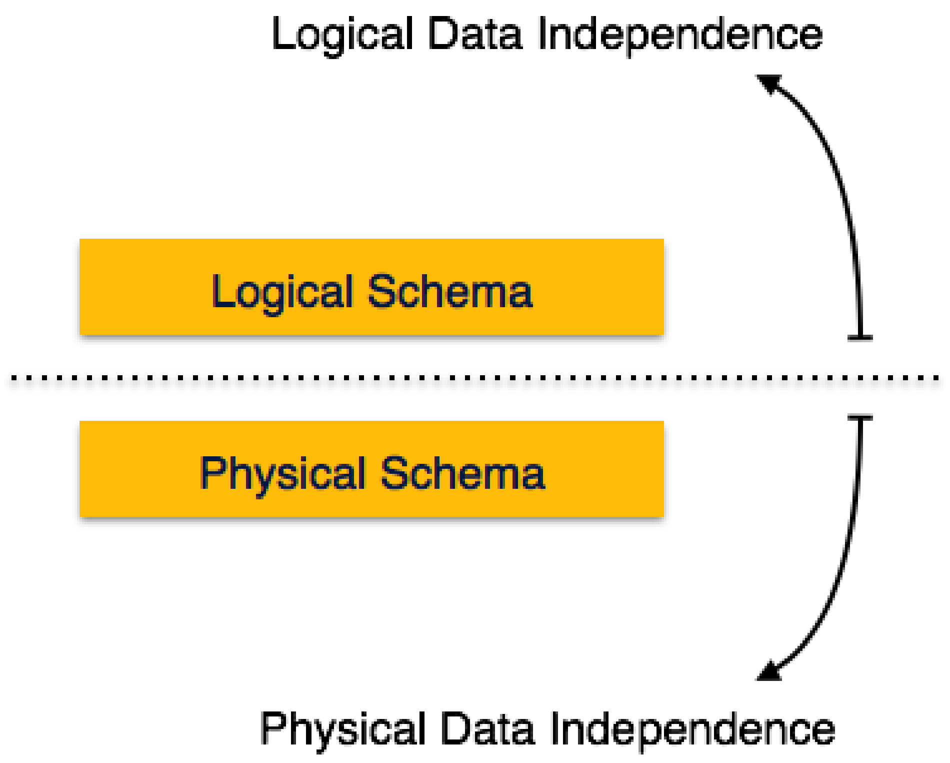
While the relational data model is very popular because of its ease of use, it requires key and index tables which drastically slows down an application. The **network** database model provides faster access to the data and is the optimal method for a fast application. So if you click on your favorite artist and see the list of their 20-plus albums and movie titles in a split second on your media player, it may just be driven by a network-model database engine under the hood. Raima Database Manager leverage both models, learn more about our advanced data modeling or download a free trial now.

CODD's rules for Relational Model

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

Data Independence

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



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

Logical Data Independence

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

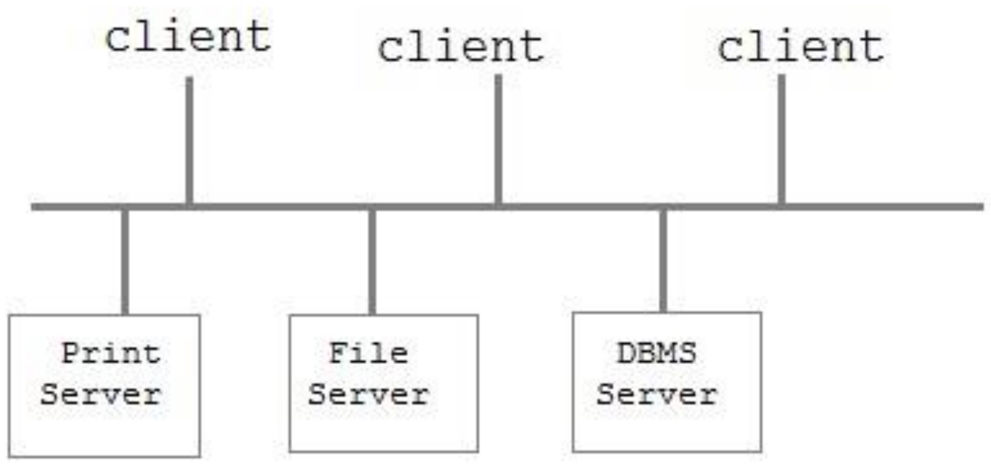
Logical data independence is a kind of mechanism, which liberalizes itself from actual data stored on the disk. If we do some changes on table format, it should not change the data residing on the disk.

Physical Data Independence

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

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

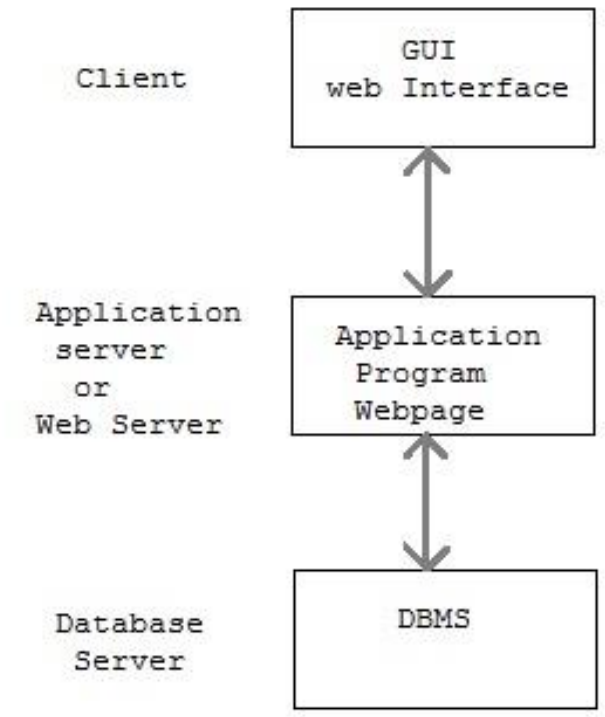
Two-tier Client / Server Architecture



Two-tier Client / Server architecture is used for User Interface program and Application Programs that runs on client side. An interface called ODBC(Open Database Connectivity) provides an API that allow client side program to call the dbms. Most DBMS vendors provide ODBC drivers. A client program may connect to several

DBMS's. In this architecture some variation of client is also possible for example in some DBMS's more functionality is transferred to the client including data dictionary, optimization etc. Such clients are called **Data server**.

Three-tier Client / Server Architecture



Three-tier Client / Server database architecture is commonly used architecture for web applications. Intermediate layer called Application server or Web Server stores the web connectivty software and the business logic(constraints) part of application used to access the right amount of data from the database server. This layer acts like medium for sending partially processed data between the database server and the client.

Codd's Rule

E.F Codd was a Computer Scientist who invented Relational model for Database management. Based on relational model, **Relation database** was created. Codd proposed 13 rules popularly known as **Codd's 12 rules** to test DBMS's concept against his relational model. Codd's rule actualy define what quality a DBMS requires in order to become a Relational Database Management System(RDBMS). Till now, there is hardly any commercial product that follows all the 13 Codd's rules. Even **Oracle** follows only eight and half out(8.5) of 13. The Codd's 12 rules are as follows.

Rule zero

This rule states that for a system to qualify as an **RDBMS**, it must be able to manage database entirely through the relational capabilities.

Rule 1 : Information rule

All information(including metadeta) is to be represented as stored data in cells of tables. The rows and columns have to be strictly unordered.

Rule 2 : Guaranted Access

Each unique piece of data(atomic value) should be accesible by : **Table Name + primary key(Row) + Attribute(column)**.

NOTE : Ability to directly access via POINTER is a violation of this rule.

Rule 3 : Systemetic treatment of NULL

Null has several meanings, it can mean missing data, not applicable or no value. It should be handled consistently. Primary key must not be null. Expression on **NULL** must give null.

Rule 4 : Active Online Catalog

Database dictionary(catalog) must have description of **Database**. Catalog to be governed by same rule as rest of the database. The same query language to be used on catalog as on application database.

Rule 5 : Powerful language

One well defined language must be there to provide all manners of access to data. Example: **SQL**. If a file supporting table can be accessed by any manner except SQL interface, then its a violation to this rule.

Rule 6 : View Updation rule

All view that are theoretically updatable should be updatable by the system.

Rule 7 : Relational Level Operation

There must be Insert, Delete, Update operations at each level of relations. Set operation like Union, Intersection and minus should also be supported

Rule 8 : Physical Data Independence

The physical storage of data should not matter to the system. If say, some file supporting table were renamed or moved from one disk to another, it should not effect the application.

Rule 9 : Logical Data Independence

If there is change in the logical structure(table structures) of the database the user view of data should not change. Say, if a table is split into two tables, a new view should give result as the join of the two tables. This rule is most difficult to satisfy.

Rule 10 : Integrity Independence

The database should be able to conforce its own integrity rather than using other programs. Key and Check constraints, trigger etc should be stored in Data Dictionary. This also make **RDBMS** independent of front-end.

Rule 11 : Distribution Independence

A database should work properly regardless of its distribution across a network. This lays foundation of distributed database.

Rule 12 : Nonsubversion rule

If low level access is allowed to a system it should not be able to subvert or bypass integrity rule to change data. This can be achieved by some sort of looking or encryption.

E-R diagram

ER Diagram is a graphical representation of **entities** and their **relationships** which helps in understanding data independent of the actual database implementation. It is mostly used for Java and other DBMS. Let us understand the terminology of **ER Modelling** through the following docket.

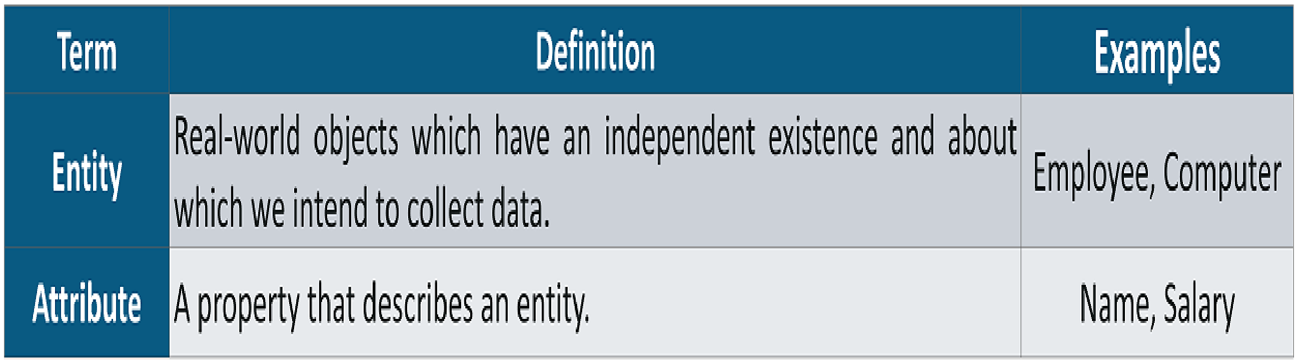
- What is an ER Diagram?
- Relationship
- The Cardinality of an ER Diagram
- The Crowfoot notations

What is an ER Diagram?

In the real world, you are often required to show the tables and their relationships, suppose you are a part of database team in your company and you are required to present the database design to business users.

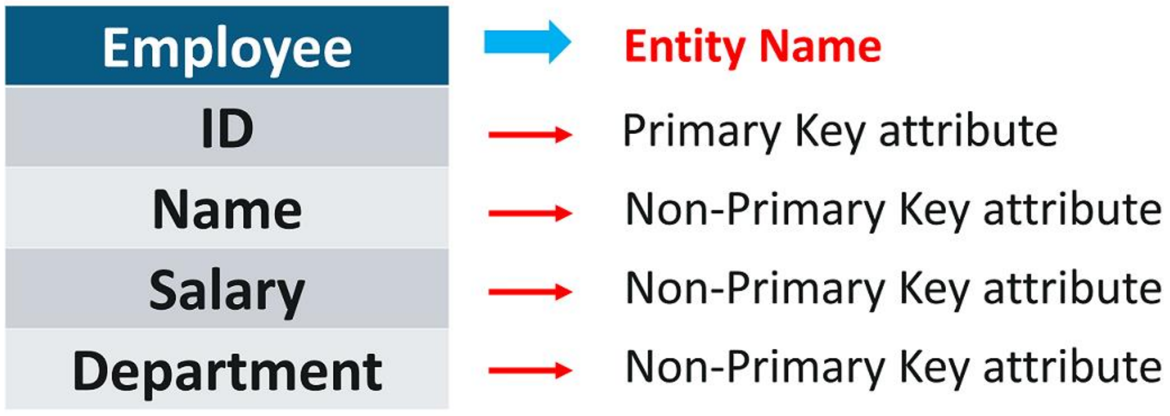
The business users are **non-technical** and it's difficult for them to read a verbose design document. What can you do? You need to use an Entity Relation (ER), Model.

The ER Diagram helps us to represent tables and their relationships in a pictorial format which would be easier to understand and more convincing to the clients and your colleagues.



Term	Definition	Examples
Entity	Real-world objects which have an independent existence and about which we intend to collect data.	Employee, Computer
Attribute	A property that describes an entity.	Name, Salary

A sample ER Diagram representing the **Employee** entity along with its attributes is presented below:


Employee
ID
Name
Salary
Department
Entity Name
Primary Key attribute
Non-Primary Key attribute
Non-Primary Key attribute
Non-Primary Key attribute

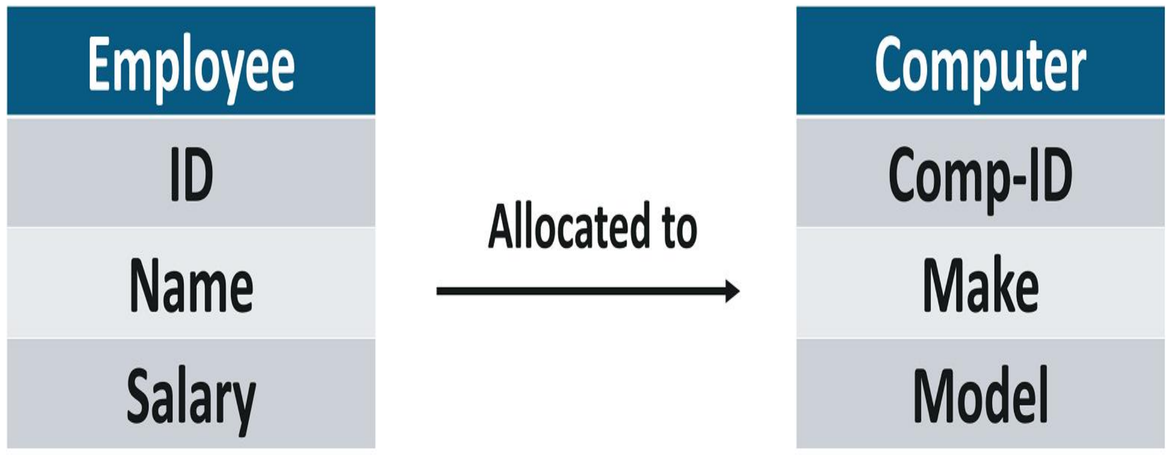
Before drawing the ER diagram, we need to understand what relationships are and how are they represented.

Relationship

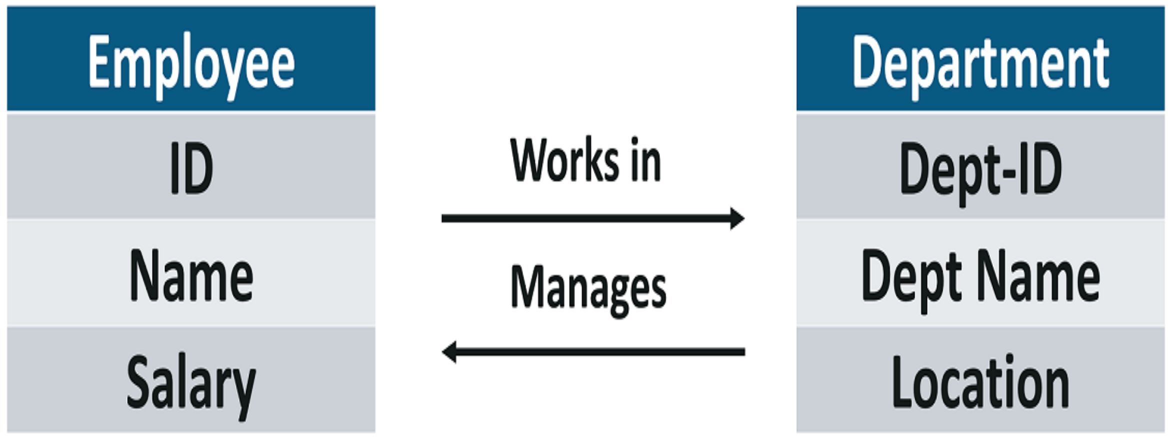
Relationships are the association of one entity with another entity. Each relationship has a name

Example:

A Computer **is allocated to** an Employee.



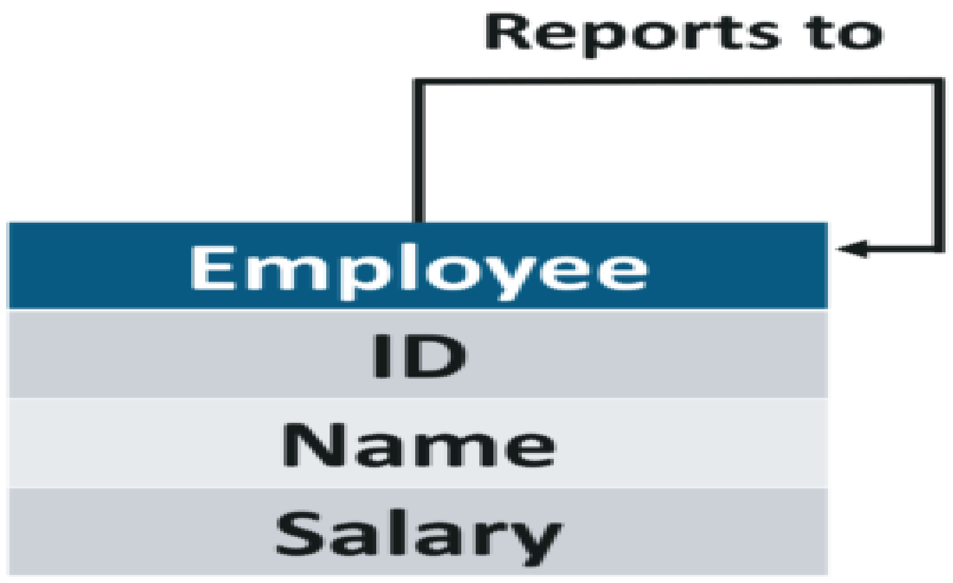
There can be more than one relationship between entities, e.g. an Employee **works in** a Department while the head of the department (also an employee) **manages** a Department.



A relationship can also exist between instances of the same entity,

Example:

An employee **reports to** another Employee.



Now, let us move into the Cardinality.

The cardinality of an ER Diagram

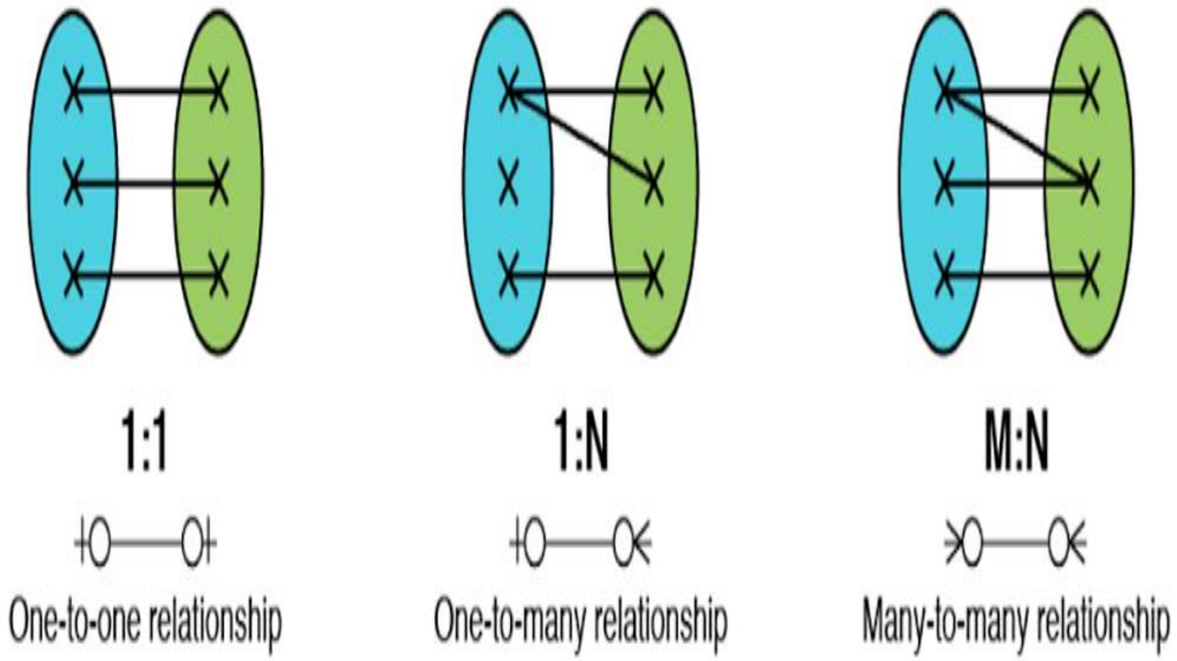
The cardinality of relationship is the number of instances in one entity which is associated with the number of instances in another.

The relationship between Employee and Computer, it helps us answer questions like how many computers can be allocated to an employee, can computers be shared between employees, can employees exist without being allocated a computer etc.

Example:

If 0 or 1 computer can be allocated to 0 or 1 employee then the cardinality of the relationship between these two entities will be 1:1.

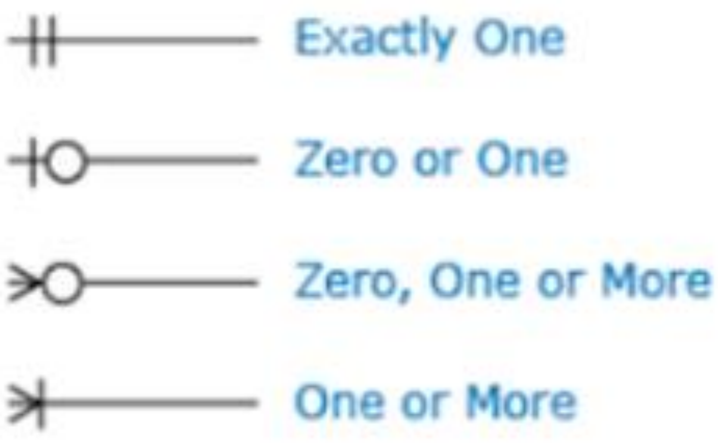
The cardinality of relationships is of three types: **1:1, 1:N** and **M:N**.



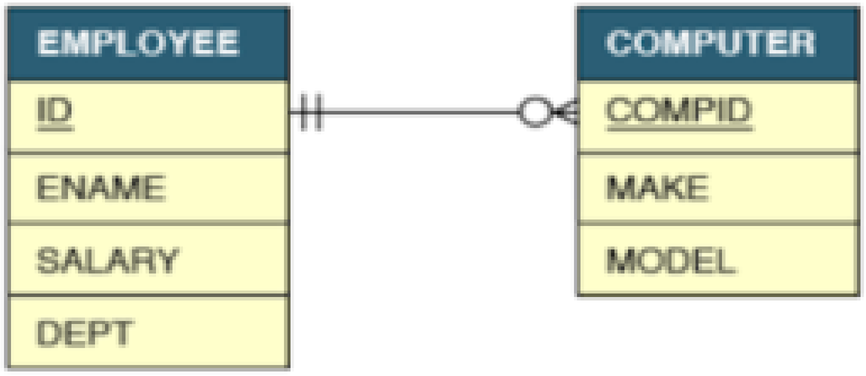
Now, let us learn the CrowFoot notations.

The Crowfoot notations

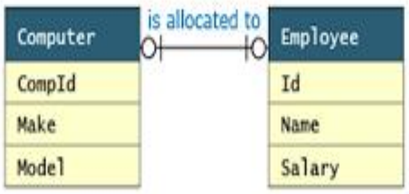
Crowfoot notation is one of the ways to represent the cardinality of the relationship in an ER Model. The notation comprises of four symbols and one of them need to be used for each entity in a relationship.

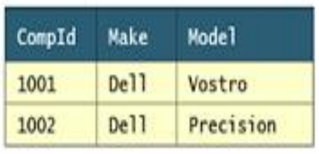


Let us say the relationship between employee and computer is such that a computer must be allocated to one and only one employee but an employee can be allocated zero or any number of computers. Such a relationship is represented by the diagram below.

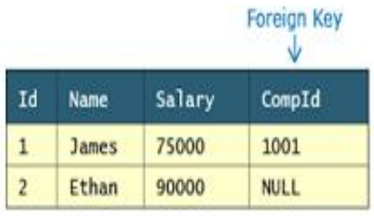


Foreign keys need to be created in tables in order to establish the relationship between entities.



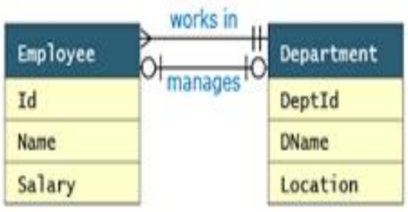


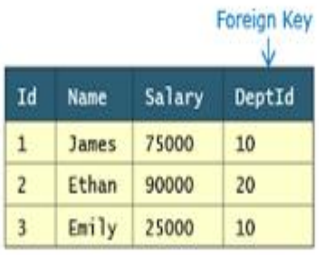
CompId	Make	Model
1001	Dell	Vostro
1002	Dell	Precision



Foreign Key

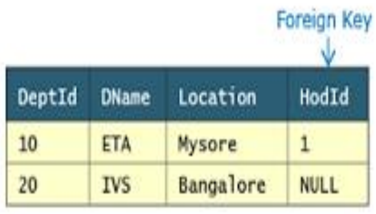
Id	Name	Salary	CompId
1	James	75000	1001
2	Ethan	90000	NULL





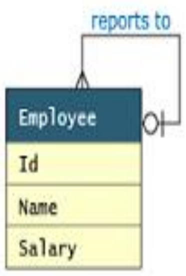
Foreign Key

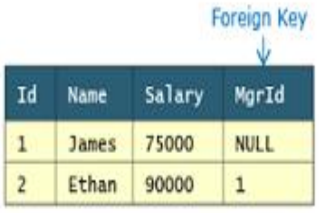
Id	Name	Salary	DeptId
1	James	75000	10
2	Ethan	90000	20
3	Emily	25000	10



Foreign Key

DeptId	DName	Location	HodId
10	ETA	Mysore	1
20	IVS	Bangalore	NULL





Foreign Key

Id	Name	Salary	MgrId
1	James	75000	NULL
2	Ethan	90000	1

Programming & Frameworks Training

JAVA CERTIFICATION TRAINING COURSE
Java Certification Training Course

Reviews

4(43240)

PYTHON SCRIPTING CERTIFICATION TRAINING
Python Scripting Certification Training

Reviews

5(9387)

NODE.JS CERTIFICATION TRAINING
Node.js Certification Training

Reviews

5(6204)

PHP & MYSQL WITH MVC FRAMEWORKS CERTIFICATION TRAINING
PHP & MySQL with MVC Frameworks Certification Training

Reviews

5(3310)

PYTHON DJANGO TRAINING AND CERTIFICATION
Python Django Training and Certification

Reviews

5(5061)

ADVANCED JAVA CERTIFICATION TRAINING
Advanced Java Certification Training

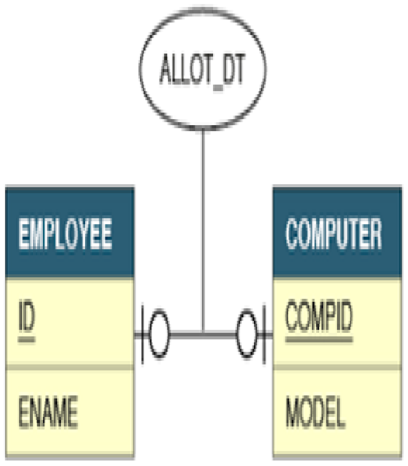
Reviews

The table in which foreign key will be created depends upon the cardinality of the relationship. Let us now discuss types of cardinalities and how it impacts foreign key creation.

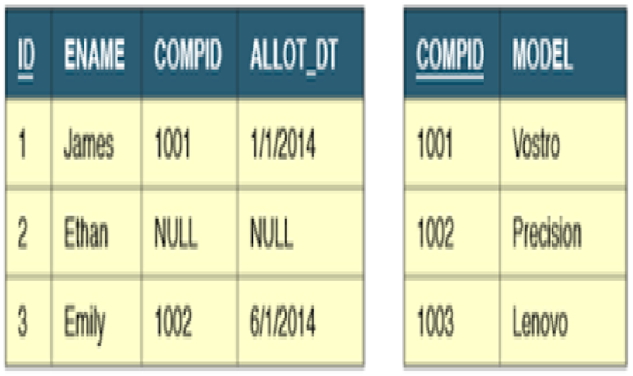
Now let’s dive straight in all these different types of relationships.

1:1 relationship

1:1 relationship represents the association between the single occurrence of one entity and a single occurrence of the second entity. For e.g. consider a company where each employee can be allocated a maximum of 1 computer and computers are not shared between employees.



ER Diagram



ID	ENAME	COMPID	ALLOT_DT
1	James	1001	1/1/2014
2	Ethan	NULL	NULL
3	Emily	1002	6/1/2014

COMPID	MODEL
1001	Vostro
1002	Precision
1003	Lenovo

Table Structure

The Allot_Dt attribute is not a property of employee **or** computer**.** It belongs to the **relationship** and is hence represented differently in the ER Model.

We can see that the employee table has two additional attributes:

- **CompId**
- **Allot_Dt**

CompId is a foreign key to establish the link between these two tables. Allot_Dt which is the attribute of the relationship is always stored in the table that has the foreign key.

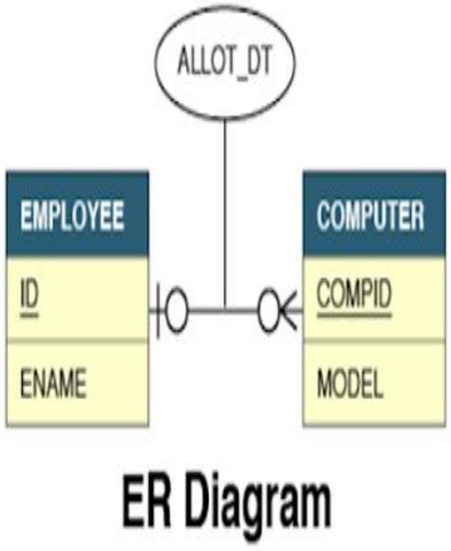
Alternatively, we could also have added Id and Allot_Dt attributes in computer table to establish the link.

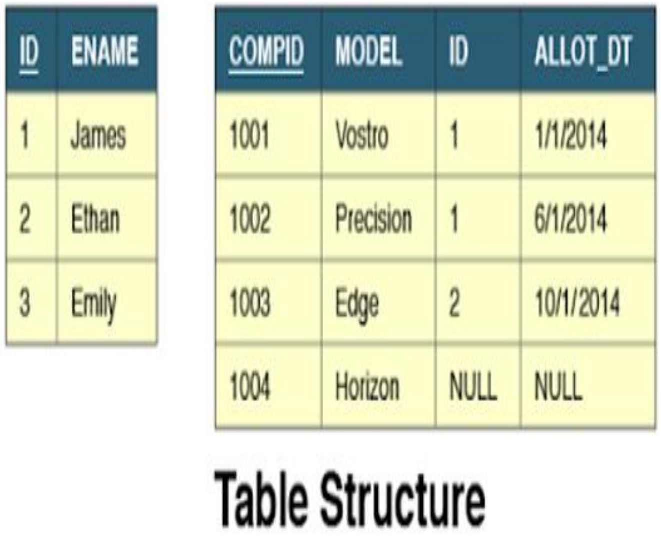
- **M:N relationship**

1 : N relationship represents the association between the single occurrence of one entity and multiple occurrences of the second entity.

Example:

Consider a company where each employee can be allocated to many computers but still, computers cannot be shared between employees.





ID	ENAME
1	James
2	Ethan
3	Emily

COMPID	MODEL	ID	ALLOT_DT
1001	Vostro	1	1/1/2014
1002	Precision	1	6/1/2014
1003	Edge	2	10/1/2014
1004	Horizon	NULL	NULL

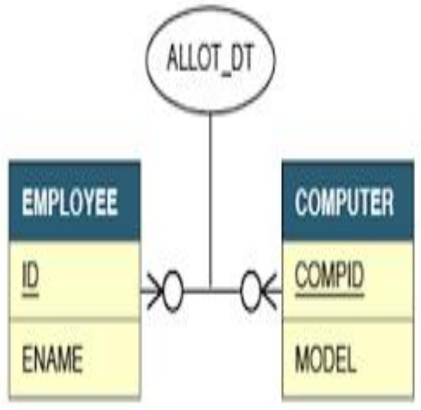
Table Structure

In **1:N** relationships, the foreign key and relationship attributes are always added to the many (N) side of the relationship. Hence these attributes are added to the Computer table. The reverse solution will not work.

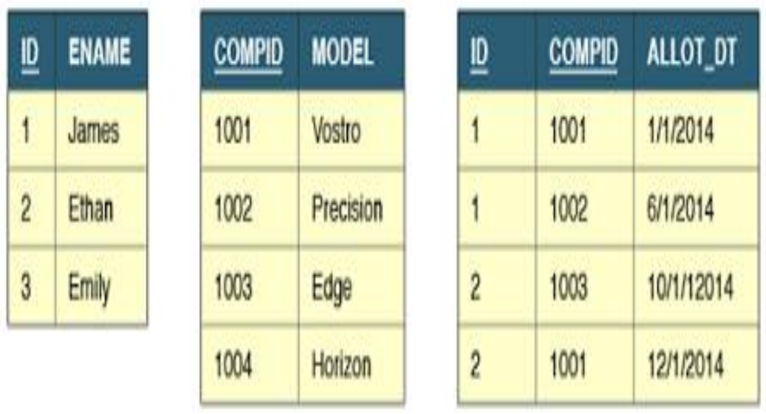
In a many to one relationship, the primary key of one entity acts as a **foreign** key on the side where many relationships are defined

- **M : N relationship**

M:N relationship represents an association between multiple occurrences of both entities. For e.g. consider a company where each employee can be allocated to many computers and computers can be shared between employees.



ER Diagram



ID	ENAME
1	James
2	Ethan
3	Emily

COMPID	MODEL
1001	Vostro
1002	Precision
1003	Edge
1004	Horizon

ID	COMPID	ALLOT_DT
1	1001	1/1/2014
1	1002	6/1/2014
2	1003	10/1/12014
2	1001	12/1/2014

Table Structure

In **M:N** relationships, the relationship is represented by a completely new table that has a composite primary key. Such a structure requires two **foreign** keys on the new table linking to the primary keys of each of the parent tables. The attribute of the relationship resides on this new table.

A many to many relationships between two entities usually results in three tables.

With this, we come to an end of this article. I hope you have understood the ER Diagram, their types, importance and their implementation through some real-time examples.

Now that you have understood the basics, check out the **Java training** by Edureka, a trusted online learning company with a network of more than 250,000 satisfied learners spread across the globe. Edureka's Java J2EE and SOA training and certification course is designed for students and professionals who want to be a Java Developer. The course is designed to give you a head start into Java programming and train you for both core and advanced Java concepts along with various Java frameworks like Hibernate & Spring.

UNIT - III

Normalization

Normalization concepts and update anomalies

In this tutorial, you will learn about data normalization in SQL. Normalization is actually a database design method that arranges the tables in a database with reduced dependency and redundancy of data. Normalization splits up the bigger tables to smaller ones and integrated them through relationships. Normalization improves data integrity. If you fail to use normalization, you could end up facing anomalies namely insertion, update, and deletion. Insertion anomalies happen to suppose if we couldn't insert data into the table without another attribute's availability. Update anomalies are actually an inconsistency in the data which could lead to data redundancy and incomplete data update. Deletion anomalies happen if you lose some attributes because of deleting other attributes.

Simply, the organization of data in the DB is called data normalization. Normalization actually demands the organization of columns and tables present in a DB to make sure that their dependants were correctly administered by the DB integrity constraints. It provides more efficiency because it splits up a bigger table to smaller ones.

Purpose of Normalization

As we all know, SQL is a language that is used to communicate with the DB. Any communication of data in the database has to be initiated and that must be normalized. Otherwise, you will end up in anomalies. It will improve data distribution as well. Normalization can be achieved by using normal forms. The normal forms we are going to learn are:

- 1 NF (First Normal Form)
- 2 NF (Second Normal Form)

- 3 NF (Third Normal Form) and
- Boyce Codd NF

Let’s see one by one with examples.

1 NF (First Normal Form)
We investigate the atomicity problem in 1 NF. In this context, atomicity implies that the values present in the table should not be divided or split up further. Simply, one cell could not carry several values. It is considered as a violation in 1 NF if a table holds a multiple value attribute. For example, have a look at the table below:

Student Admission No	Student Name	Mobile Number	Outstanding Fees
1PRI001	Aravindan	-9678900476	25,000
		-9556678854	
1PRI002	Darshan	-9887765341	1,000
1PRI003	Saravanan	-9443356698	33,000
1PRI004	Ramkumar	-6345678810	50,000
		-8667890476	

Evidently, you can notice that the phone number column contains more than one value and thus, it is a violation in 1 NF. If we apply 1 NF, the table will automatically get normalized (arranged) like as follows:

Student admission no	Student Name	Mobile Number	Outstanding Fees
1PRI001	Aravindan	-9678900476	25,000
1PRI001	Aravindan	-9556678854	25,000
1PRI002	Darshan	-9887765341	1,000
1PRI003	Saravanan	-9443356698	33,000
1PRI004	Ramkumar	-6345678810	50,000
1PRI004	Ramkumar	-8667890476	50,000

As per the above table, you could visualize every column with distinct values and thus we achieved atomicity using 1 NF.

Click Here – Get SQL Training with Real-Time Projects

2NF (Second Normal Form)

In the case of 2 NF, the basic need for satisfying 2 NF is that the table must be present in 1 NF and there should not be any partial dependency, which means the actual subset of the candidate key decides the attribute which is non-prime. Let's look at an example to understand 2 NF better!

Student admission no	Class Room number	Classroom Name
1PRI001	South-A1	Blackberries
1SEC001	South-A4	Avocado

Student admission no	Class Room number	Classroom Name
2PRI001	South-A2	Jingle bells
2SEC001	South-A5	Craneberries

normalized (arranged) as follows:

The above table contains a composite primary key namely Student admission number and Classroom number. Here, Classroom location is a non-key attribute evidently. This Classroom location will depend on the Classroom number, which is actually a part of the primary key. Thus, the above table is a violation of 2 NF. In order to change the above table to 2 NF, we have to divide the table into two portions as follows:

Student admission no	Class Room number
1PRI001	South-A1
1SEC001	South-A4
2PRI001	South-A2
2SEC001	South-A5

Student admission no	Class Room number
1PRI001	South-A1
1SEC001	South-A4
2PRI001	South-A2
2SEC001	South-A5

I hope, you could visualize that the partial dependency has been removed in the second table by applying 2 NF. So, the column Class Room Name entirely depends on the table's primary key, i.e Class Room Number.

3NF (Third Normal Form)

In the case of 3 NF, it follows the same way that 2 NF functions. Here, the table must be present in 2 NF before working with 3 NF. Also, a transitive dependency is not allowed in 3 NF for non-prime attributes. This implies that the non-prime attributes which do not contain a candidate key will not depend on the rest of the non-prime attributes in a table. We can conclude transitive dependency is an indirect functional dependency, i.e A→C (which means A determines C) in which A→B and B→C (but the inverse is not valid i.e B→A is invalid) Let's get a clear understanding of 3 NF with the following example:

Employee ID	Employee Name	Department ID	Department	Location
1SW15TE01	Sarath	15TE01	Testing	Hyderabad
1SW15BE01	Ramesh	15BE01	SQL	Chennai

Employee ID	Employee Name	Department ID	Department	Location
1SW15DE01	Raj	15DE01	Dotnet	Kochi
1SW15DE02	Kumar	15DE02	Java	Bengaluru

Looking at the above table, we can understand that the Employee ID determines Department ID and Department ID determines the department. Thus, Employee ID determines Department via Department ID. This proves that we accomplished transitive function dependency. But, the above structure violates 3 NF because it does not satisfy the rules of 3 NF. So, we have to divide the tables as below:

Employee ID	Employee Name	Department ID	Location
1SW15TE01	Sarath	15TE01	Hyderabad
1SW15BE01	Ramesh	15BE01	Chennai
1SW15DE01	Raj	15DE01	Kochi
1SW15DE02	Kumar	15DE02	Bengaluru

From the above tables, you could visualize that the entire non-key attributes become

Department ID	Department
15TE01	Testing
15BE01	SQL
15DE01	Dotnet
15DE02	Java

completely dependent on the primary key. As in the first table, Employee Name, Department ID and Location depends on Employee ID, whereas in the second table, the Department depends on Department ID.

Boyce Codd NF (BCNF)

BCNF is also called as 3.5 NF because it is an upgrade of 3 NF. Two researchers Boyce and Codd developed this BCNF concept so as to address some particular anomalies that that doesn't fall under the 3 NF category. Like other NF techniques, BCNF also has certain conditions to be satisfied. First, BCNF should satisfy 3 NF. In the case of BCNF, if each and every functional dependency, X → Y, then, X will act as the Super key of that specific table.

For example, have a look at the table below:

Stud ID	Course of Study	Name of the Professor
1SD17SW01	Java	Magesh
1SD17SW02	Dotnet	Karthik
1SD17SW03	C++	Praba

Stud ID	Course of Study	Name of the Professor
1SD17SW04	Dotnet	Ramesh
1SD17SW05	SQL	Lokesh

As per the above table, we can clarify the following:

- Any student can select multiple subjects of study
- You can have multiple teachers to teach one particular subject.
- For every subject, a teacher has to allocated to the student.

In the above table, except for the BCNF, all other NF techniques were satisfied. Let's discuss the reason of it. Stud ID and Course of Study provides the primary key. This implies that the Course of Study column is actually a prime attribute. We could see yet another dependency here, i.e Name of the Professor→ Course of Study.
Here, Course of Study is actually a prime attribute whereas the Name of the Professor is a nonprime attribute, which is actually a violation of BCNF. Therefore, to achieve BCNF, we have to separate the table into two portions as Stud ID which is there already and another new column named Prof ID.

Stud ID	Prof ID
1SD17SW01	1PF17SW01
1SD17SW02	1PF17SW02
1SD17SW03	1PF17SW03
1SD17SW04	1PF17SW04
1SD17SW05	1PF17SW05

In the second table, Prof ID, Name of the Professor and Course of Study will be present.

Prof ID	Name of the Professor	Course of Study
1PF17SW01	Magesh	Java
1PF17SW02	Karthik	Dotnet
1PF17SW03	Praba	C++
1PF17SW04	Ramesh	Dotnet
1PF17SW05	Lokesh	SQL

With this, we achieved BCNF. We thus conclude this tutorial about Normalization in SQL. I hope you got a better understanding!

Functional dependencies

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

X ———> Y

The left side of the above FD diagram is called the *determinant*, and the right side is the *dependent*. Here are a few examples.

In the first example, below, SIN determines Name, Address and Birthdate. Given SIN, we can determine any of the other attributes within the table.

SIN ———-> Name, Address, Birthdate

For the second example, SIN and Course determine the date completed (DateCompleted). This must also work for a composite PK.

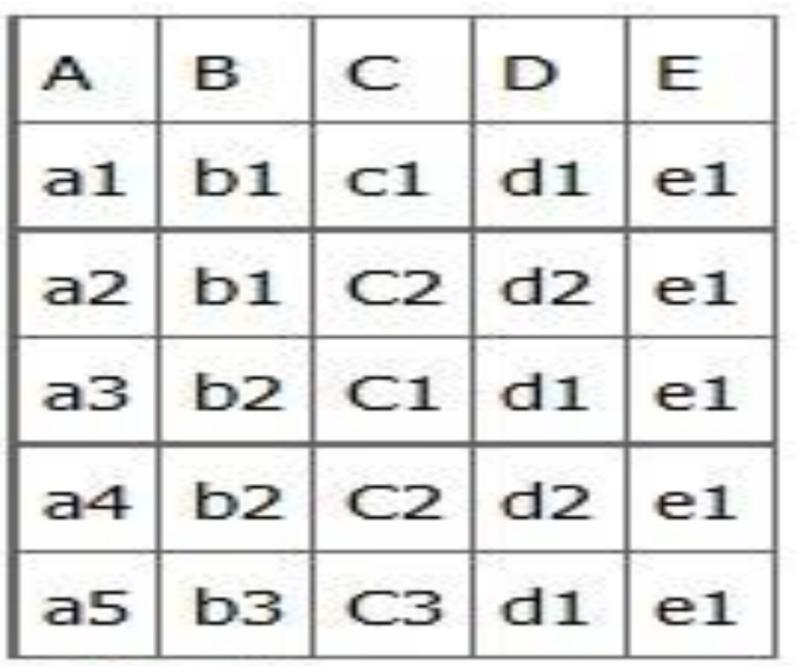
SIN, Course ———> DateCompleted

The third example indicates that ISBN determines Title.

ISBN ———> Title

Rules of Functional Dependencies

Consider the following table of data r(R) of the relation schema R(ABCDE) shown in Table 11.1.



A	B	C	D	E
a1	b1	c1	d1	e1
a2	b1	C2	d2	e1
a3	b2	C1	d1	e1
a4	b2	C2	d2	e1
a5	b3	C3	d1	e1

Table R

As you look at this table, ask yourself: What kind of dependencies can we observe among the attributes in Table R? Since the values of A are unique (a1, a2, a3, etc.), it follows from the FD definition that:

A → B,   A → C,   A → D,   A → E

- It also follows that A →BC (or any other subset of ABCDE).
- This can be summarized as   A →BCDE.
- From our understanding of primary keys, A is a primary key.

Since the values of E are always the same (all e1), it follows that:

A → E,   B → E,   C → E,   D → E

However, we cannot generally summarize the above with ABCD → E because, in general,   A → E,   B → E,   AB → E.

Other observations:

1. Combinations of BC are unique, therefore BC → ADE.
2. Combinations of BD are unique, therefore BD → ACE.
3. If C values match, so do D values.

1. Therefore, C → D
2. However, D values don't determine C values
3. So C does not determine D, and D does not determine C.

Looking at actual data can help clarify which attributes are dependent and which are determinants.

Inference Rules

Armstrong's axioms are a set of inference rules used to infer all the functional dependencies on a relational database. They were developed by William W. Armstrong. The following describes what will be used, in terms of notation, to explain these axioms.

Let R(U) be a relation scheme over the set of attributes U. We will use the letters X, Y, Z to represent any subset of and, for short, the union of two sets of attributes, instead of the usual X U Y.

Axiom of reflexivity

This axiom says, if Y is a subset of X, then X determines Y (see Figure 11.1).

$$\text{If } Y \subseteq X \text{, then } X \rightarrow Y$$

For example, **PartNo —> NT123** where X (PartNo) is composed of more than one piece of information; i.e., Y (NT) and partID (123).

Axiom of augmentation

The axiom of augmentation, also known as a partial dependency, says if X determines Y, then XZ determines YZ for any Z (see Figure 11.2).

$$\text{If } X \rightarrow Y\text{, then } XZ \rightarrow YZ \text{ for any } Z$$

The axiom of augmentation says that every non-key attribute must be fully dependent on the PK. In the example shown below, StudentName, Address, City, Prov, and PC (postal code) are only dependent on the StudentNo, not on the StudentNo and Grade.

StudentNo, Course —> StudentName, Address, City, Prov, PC, Grade, DateCompleted

This situation is not desirable because every non-key attribute has to be fully dependent on the PK. In this situation, student information is only partially dependent on the PK (StudentNo).

To fix this problem, we need to break the original table down into two as follows:

- Table 1: StudentNo, Course, Grade, DateCompleted
- Table 2: StudentNo, StudentName, Address, City, Prov, PC

Axiom of transitivity

The axiom of transitivity says if X determines Y, and Y determines Z, then X must also determine Z (see Figure 11.3).

The table below has information not directly related to the student; for instance, ProgramID and ProgramName should have a table of its own. ProgramName is not dependent on StudentNo; it's dependent on ProgramID.

StudentNo —> StudentName, Address, City, Prov, PC, ProgramID, ProgramName

This situation is not desirable because a non-key attribute (ProgramName) depends on another non-key attribute (ProgramID).

To fix this problem, we need to break this table into two: one to hold information about the student and the other to hold information about the program.

- Table 1: StudentNo —> StudentName, Address, City, Prov, PC, ProgramID
- Table 2: ProgramID —> ProgramName

However we still need to leave an FK in the student table so that we can identify which program the student is enrolled in.

Union

This rule suggests that if two tables are separate, and the PK is the same, you may want to consider putting them together. It states that if X determines Y and X determines Z then X must also determine Y and Z (see Figure 11.4).

For example, if:

- SIN —> EmpName
- SIN —> SpouseName

You may want to join these two tables into one as follows:

SIN –> EmpName, SpouseName

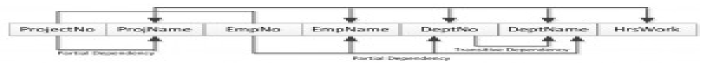
Some database administrators (DBA) might choose to keep these tables separated for a couple of reasons. One, each table describes a different entity so the entities should be kept apart. Two, if SpouseName is to be left NULL most of the time, there is no need to include it in the same table as EmpName.

Decomposition

Decomposition is the reverse of the Union rule. If you have a table that appears to contain two entities that are determined by the same PK, consider breaking them up into two tables. This rule states that if X determines Y and Z, then X determines Y and X determines Z separately (see Figure 11.5).

Dependency Diagram

A dependency diagram, shown in Figure 11.6, illustrates the various dependencies that might exist in a *non-normalized table*. A non-normalized table is one that has data redundancy in it.



The following dependencies are identified in this table:

- ProjectNo and EmpNo, combined, are the PK.
- Partial Dependencies:

- ProjectNo —> ProjName
- EmpNo —> EmpName, DeptNo,

- ProjectNo, EmpNo —> HrsWork

Transitive Dependency:

- DeptNo —> DeptName

Key Terms

Armstrong's axioms: a set of inference rules used to infer all the functional dependencies on a relational databaseDBA: database administrator

decomposition: a rule that suggests if you have a table that appears to contain two entities that are determined by the same PK, consider breaking them up into two tables

dependent: the right side of the functional dependency diagram

determinant: the left side of the functional dependency diagram

functional dependency (FD): a relationship between two attributes, typically between the PK and other non-key attributes within a table

non-normalized table: a table that has data redundancy in it

Union: a rule that suggests that if two tables are separate, and the PK is the same, consider putting them together

Exercises

See Chapter 12.

Attributions

This chapter of Database Design (including images, except as otherwise noted) is a derivative copy of Armstrong's axioms by Wikipedia the Free Encyclopedia licensed under Creative Commons Attribution-ShareAlike 3.0 Unported

The following material was written by Adrienne Watt:

1. some of Rules of Functional Dependencies
2. Key Terms

Multivalued and join dependencies:-

Multivalued

When existence of one or more rows in a table implies one or more other rows in the same table, then the Multi-valued dependencies occur.

If a table has attributes P, Q and R, then Q and R are multi-valued facts of P.

It is represented by double arrow −

```
->->
```

For our example:

```
P->->Q
P->->R
```

In the above case, Multivalued Dependency exists only if Q and R are independent attributes.

A table with multivalued dependency violates the 4NF.

Example

Let us see an example &mins;

<Student>

StudentName	**CourseDiscipline**	**Activities**
Amit	Mathematics	Singing
Amit	Mathematics	Dancing
Yuvraj	Computers	Cricket
Akash	Literature	Dancing

Akash	Literature	Cricket
Akash	Literature	Singing

In the above table, we can see Students **Amit** and **Akash** have interest in more than one activity.

This is multivalued dependency because **CourseDiscipline** of a student are independent of Activities, but are dependent on the student.

Therefore, multivalued dependency −

StudentName ->-> CourseDiscipline
StudentName ->-> Activities

The above relation violates Fourth Normal Form in Normalization.

To correct it, divide the table into two separate tables and break Multivalued Dependency −

<StudentCourse>

StudentName	**CourseDiscipline**
Amit	Mathematics
Amit	Mathematics
Yuvraj	Computers
Akash	Literature
Akash	Literature
Akash	Literature

<StudentActivities>

StudentName	**Activities**
Amit	Singing
Amit	Dancing
Yuvraj	Cricket
Akash	Dancing
Akash	Cricket
Akash	Singing

This breaks the multivalued dependency and now we have two functional dependencies −

StudentName -> CourseDiscipline
StudentName - > Activities

Join dependency

If a table can be recreated by joining multiple tables and each of this table have a subset of the attributes of the table, then the table is in Join Dependency. It is a generalization of Multivalued Dependency

Join Dependency can be related to 5NF, wherein a relation is in 5NF, only if it is already in 4NF and it cannot be decomposed further.

Example

<Employee>

EmpName	**EmpSkills**	**EmpJob** (Assigned Work)
Tom	Networking	EJ001
Harry	Web Development	EJ002
Katie	Programming	EJ002

The above table can be decomposed into the following three tables; therefore it is not in 5NF:

<EmployeeSkills>

EmpName	**EmpSkills**
Tom	Networking
Harry	Web Development
Katie	Programming

<EmployeeJob>

EmpName	**EmpJob**
Tom	EJ001
Harry	EJ002
Katie	EJ002

<JobSkills>

EmpSkills	**EmpJob**
Networking	EJ001
Web Development	EJ002
Programming	EJ002

Our Join Dependency −

{(EmpName, EmpSkills), (EmpName, EmpJob), (EmpSkills, EmpJob)}

The above relations have join dependency, so they are not in 5NF. That would mean that a join relation of the above three relations is equal to our original relation **<Employee>**.

Normal Forms: (1 NF, 2 NF, 3NF, BCNF, 4NF, and 5NF)

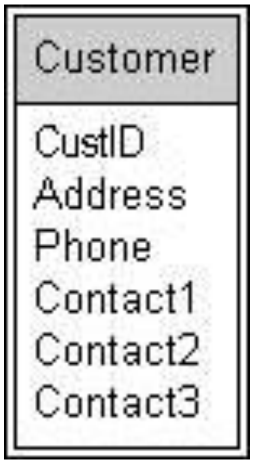
The next sections of this paper will describe each of the normal forms and how they are applied. There will be examples used to describe the form and its application. The examples chosen are obviously wrong and are designed to clearly demonstrate the normal form being discussed.

In your actual design work the normalization problems will probably be more subtle and require a much more careful study to discover and repair.

1st Normal Form (1NF)

Reduce entities to first normal form (1NF) by removing repeating or multi-valued attributes to another, child entity.

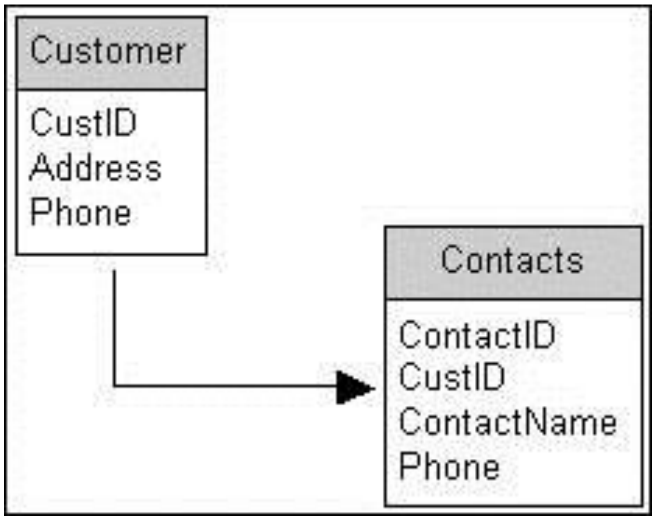
To understand 1st Normal Form we will use the table design below.



To discover the problem in this design we must consider the domains for the fields in the table. The CustID is defined as the customer Primary key ID, the Name is the name of the customer, Contact1 is the name of a contact person, Contact2 is the name of a contact person, and Contact3 is the name of a contact person.

The fact that Contact1, 1, and 3 all have the same domain definition proves that in fact there is only one attribute, contact person, and that we need multiple values for that attribute. This is a multi-valued attribute.

The 1st NF design for this situation is shown below.



Notice the creation of the new entity for Contacts and the relation of that entity to the original Customer entity. Using this new design the customer can have any number of contacts from none to the capacity of the table storing the contact names.

What about the client who tells us that their customer will never have more than three contact names? Do we really need to do this for those situations?

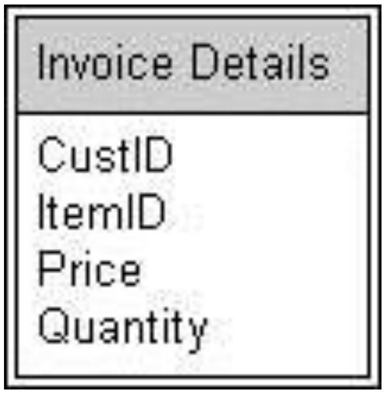
Well, reread what I said earlier about clients and the word never. Besides that, if we provide the three fields for contact names and most customers have only one name, we are wasting a lot of space. For a contact name of 40 characters and 1 million customer records that would amount to approximately 40 MB of wasted space.

Also, the first customer that comes along with four or more contact names would require that the user either use two customer records, not store all of the contact names, or pay for a revision to the data design to allow the fourth name. With the 1st Normal Form structure none of these things are an issue. If the customer has only one contact then there is only one record in the Contacts table. If the customer has 300 contact names, then there are 300 records in the contacts table.

Reduce entities in 1NF to 2NF by removing attributes that are not dependent on the whole primary key.

2nd Normal Form (2NF)

The figure below will be used to study this normal form.



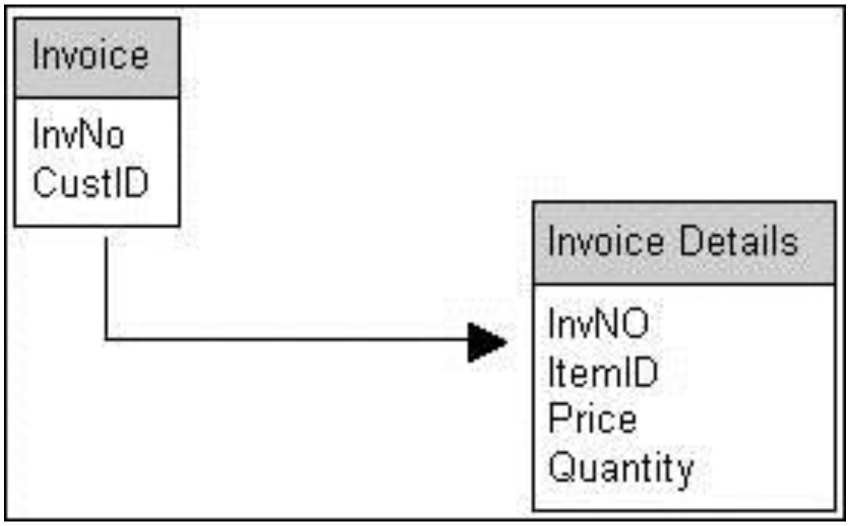
Invoice Details
CustID
ItemID
Price
Quantity

The primary key for the invoice details table in the figure is the combination of InvNo and LineNo. The two fields together comprise the primary key. 2nd NF deals with non-key attributes that are not dependent on the entire primary key but rather only on part of it.

The ItemID and Price Quantity are dependent on the whole primary key. You cannot know the item sold or its quantity price break without knowing the invoice and which line of the invoice you are interested in.

However the CustID will remain the same for all lines on an invoice. This means that CustID is dependent on the InvNo only and not ion the LineNo. CustID is dependent on part of the primary key.

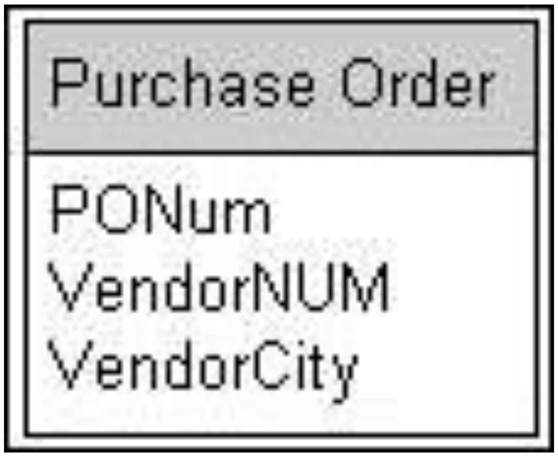
To fix this we move the CustID field to another table where it is dependent on the whole primary key.



3rd Normal Form (3NF)

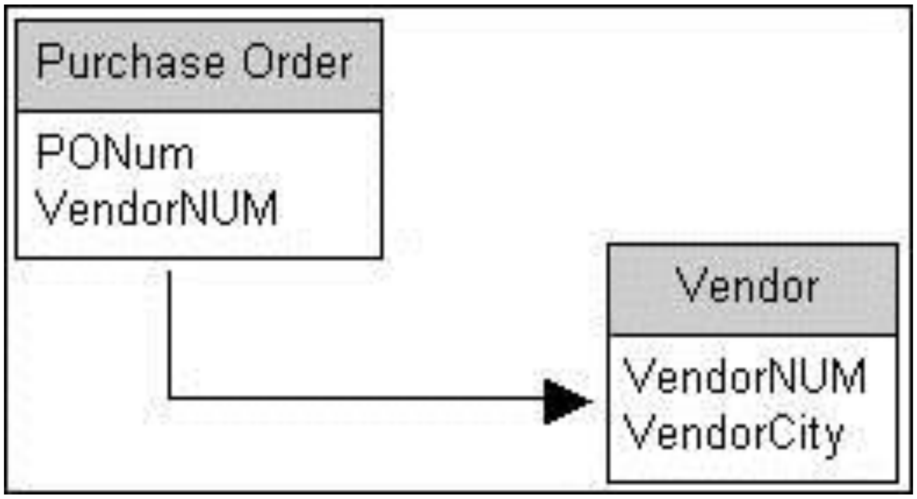
Reduce entities in 2NF to 3NF by removing attributes that depend on other, non-key attributes (other than alternate keys).

The golden rule of relational databases is, "the key, the whole key, and nothing but the key". The 3rd normal form deals with attributes that are codependent on the primary key and another, non-key, attribute. The figure below shows a table design that violates the 3rd normal form.



With the 3rd normal form we are trying to identify non-key attributes that have a dependency on other non-key attributes (other than alternate keys). In figure 13 the there are four non-key attributes that are all dependent on the primary key, that is to know the VendorID, VendorCity, Date, or Terms of a purchase order you must know which purchase order you are looking at. However the VendorCity is also dependent on the VendorID for its value. That is if you change the VendorID on a purchase order the VendorCity will also need to change.

The solution for this example is shown in below.



We have moved the VendorCity out of the purchase order table and put it in the Vendor table where the VendorID is the primary key.

Perhaps you have heard someone say that it is not a good design, in a relational database, to store the results of a calculation in a table. Why not? What rule does this break? It violates 3rd normal form.

If I have a table for invoice detail lines and it has a UnitPrice field, a quantity field, and a TotalPrice field (which is calculated by multiplying the UnitPrice by the Quantity) then I have at least one field that is codependent, the TotalPrice field. The TotalPrice for a line is dependent on the line number, but it is also dependent on both the UnitPrice and the Quantity. If either UnitPrice or Quantity changes then the TotalPrice will also need to change.

Y) Is 3rd Normal Form good enough?

I have often heard people say that 3rd normal form is good enough; perhaps you have too. Is this true? Is 3rd normal form good enough? Well, I would have to ask that if 3rd normal form was as far as it is necessary to go with normalization then why are there three more normal forms after 3rd?

n truth, the next three normal forms only apply in certain specific situations and if none of those situations exist in the data design, then 3rd normal form is 5th normal form an fully normalized.

Z) Boyce-Codd Normal Form (BCNF)

Reduce entities in 3NF to BCNF by ensuring that they are in 3NF for any feasible choice of candidate key as primary key.

The next normal form is named after the two people who first described it, Boyce and Codd. This normal form is only required for tables that have more than one candidate for the primary key. The rule is simple; if the table is in 3rd normal form for the primary key being used, insure that it is also in 3rd normal form for any of the alternate keys as well.

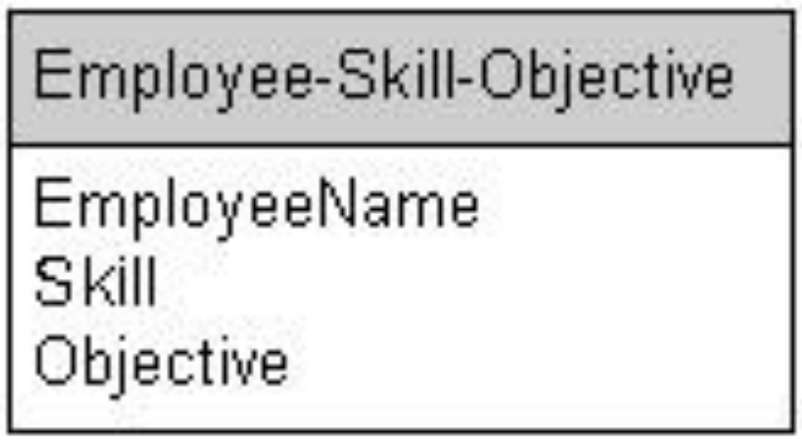
Imagine an employee table that has attributes for Social Security Number, Employee Clock Number, and Employee ID (a surrogate primary key). 3rd normal form would apply the first three rules using the Employee ID as the primary key. Boyce-Codd normal form would go back and apply the first three rules using the Social Security Number and then using the Employee Clock Number as the primary key. When the table structure is in 3rd normal form no matter which candidate for primary key is used, then it is in Boyce-Codd normal form.

4th Normal Form (4NF)

Reduce entities in BCNF to 4NF by removing any independently multi-valued components of the primary key to multiple new parent entities.

4th normal form is only applicable when the primary key is comprised of two or more attributes. With a primary key of only one attribute there is no need to check 4th normal form. 4th and 5th normal forms resolve problems within the primary key itself.

In figure 15 we have a design that is meant to record and track employees, their skills, and their objectives. The primary key for the table is the combination of the Employee ID, the Skill ID, and the Objective ID. The problem with this design is the independence of the skill and objective attributes comprising the primary key.



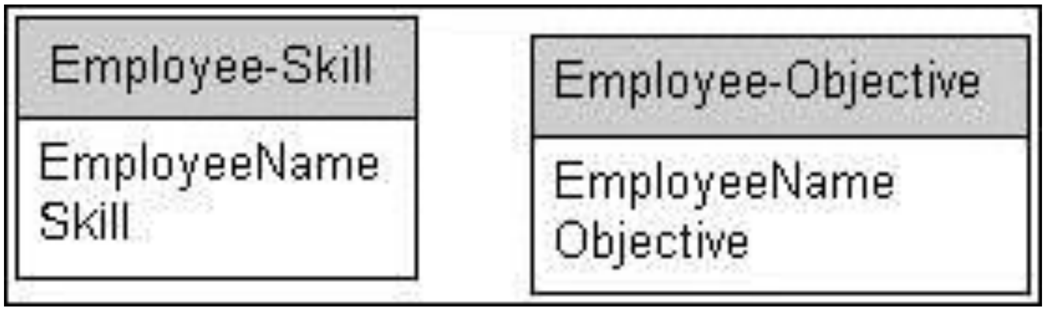
To really understand the nature of the problem, let's consider some data from this table:

EmpID	Skill	Objective
Jones	Accounting	More Money
Jones	Accounting	Master's Degree

Jones	Public Speaking	More Money
Jones	Public Speaking	Master’s Degree

Looking at the sample data, what would need to happen if Jones was to tell you he had an objective of getting a doctorate degree too? How many record would you need to ad for that change? What if he received his Masters Degree? Again how many records would need to change? Both situations require that more than one record change in order to record the change in the data.

Below is shown the same information being recorded, but the design is in 4th normal form. Any of the events asked about in the previous paragraph will only involve one record in the new design.

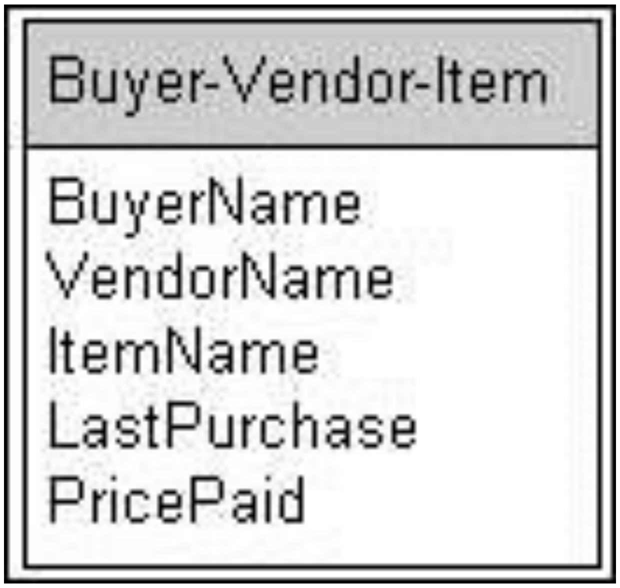


5th Normal Form (5NF)

Reduce entities in 4NF to 5NF by removing pair-wise cyclic dependencies (appearing within composite primary keys with three or more component attributes) to three or more new parent entities.

The 5th normal form is another one that is only required when the primary key has more than one attribute. In fact, with 5th normal form the primary key must use three or more attributes.

Reading the definition for this normal form can be stress inducing for sure. If you take it apart and understand each piece separately it really isn’t that complex. The definition refers to pair-wise cyclic dependencies. Pair-wise means taking two attributes at a time, dependencies is referring to the value of one attribute being dependent on the value of another. The cyclic is simply saying that in a primary key of three attributes you need the value of the other two to determine the value of any one of them. The figure below shows an example of a 5th normal form problem.


Buyer-Vendor-Item

BuyerName
VendorName
ItemName
LastPurchase
PricePaid

This design is to record information about a retail buying operation. The requirement is to track the buyers, from whom do they buy, and what do they buy. The table design has the combination of Buyer, Vendor, and Item as the primary key.

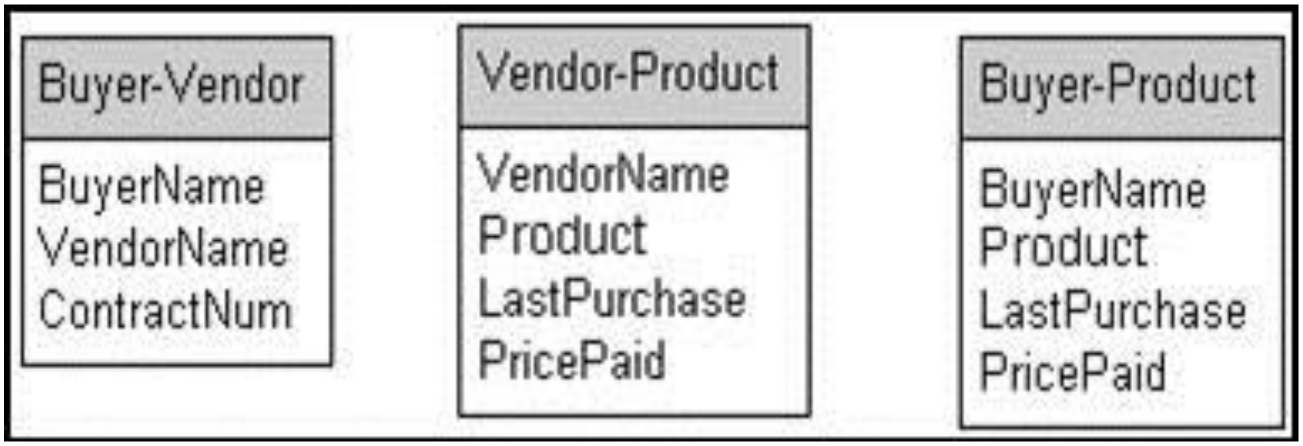
If you analyze the relationship between the components of the primary key in this design you will realize that if you want to know the buyer, you must first determine the vendor and item. If you want to know the vendor, you need the buyer and item. Finally if you want the item, you must know the vendor and buyer. Notice the pair wise (you always need to know two) cyclic (no matter which one you need it is the other two that it depends on) dependency.

To appreciate the nature of the difficulty having a table that is in violation of 5th normal form will present to you, consider the following sample data.

Buyer	Vendor	Item
Mary	Jordache	Jeans
Mary	Jordache	Sneakers
Sally	Jordache	Jeans
Mary	Liz Claiborne	Blouses
Sally	Liz Claiborne	Blouses

Like 4th normal form, the major problem areas with 5th normal form have to do with data updates. For example, if Liz Claiborne were to introduce a new line of Jeans, how many records would need to be added to this table to reflect that change? Two, since both Mary and Sally buy from Claiborne and both Mary and Sally buy Jeans. What if Jordache dropped their line of jeans? Again, two records need to be modified (actually deleted) to reflect this change.

Below is the design reduced to the 5th normal form.


Buyer-Vendor
BuyerName
VendorName
ContractNum
Vendor-Product
VendorName
Product
LastPurchase
PricePaid
Buyer-Product
BuyerName
Product
LastPurchase
PricePaid

UNIT - IV

SQL

SQL Constructs

SQL Tutorial of w3resource aims to meet the need of a beginner to learn SQL without any prior experience. Having said that, it by no means superficial. On the contrary, it offers all the material one needs to successfully build a database and write SQL queries ranging from a one liner like "SELECT * FROM table_name" to fairly non-trivial ones taking multiple tables in the account.

At the outset, we need to tell you, this SQL Tutorial adheres to SQL:2003 standard of ANSI. This is important because if you are learning something as important as SQL, there is no point learning if you don't know which version or standard you are studying. We have diligently added as many features as possible while creating this SQL Tutorial. There is Syntax, Query, Explanation of a query and pictorial presentation to help you understand concepts better. On top of these, we have hundreds of Exercises with an online editor, quizzes. So you may practice concepts and queries without leaving your browser.

Contents:

Introduction

In June 1970 Dr. E. F. Codd published the paper, "A Relational Model of Data for Large Shared Data Banks" in the Association of Computer Machinery (ACM) journal. Codd's model is now accepted as the definitive model for relational database management systems (RDBMS).

Using Codd's model the language, Structured English Query Language (SEQUEL) was developed by IBM Corporation in San Jose Research Center. The language was first called SEQUEL but Official pronunciation of SQL is ESS QUE ELL.

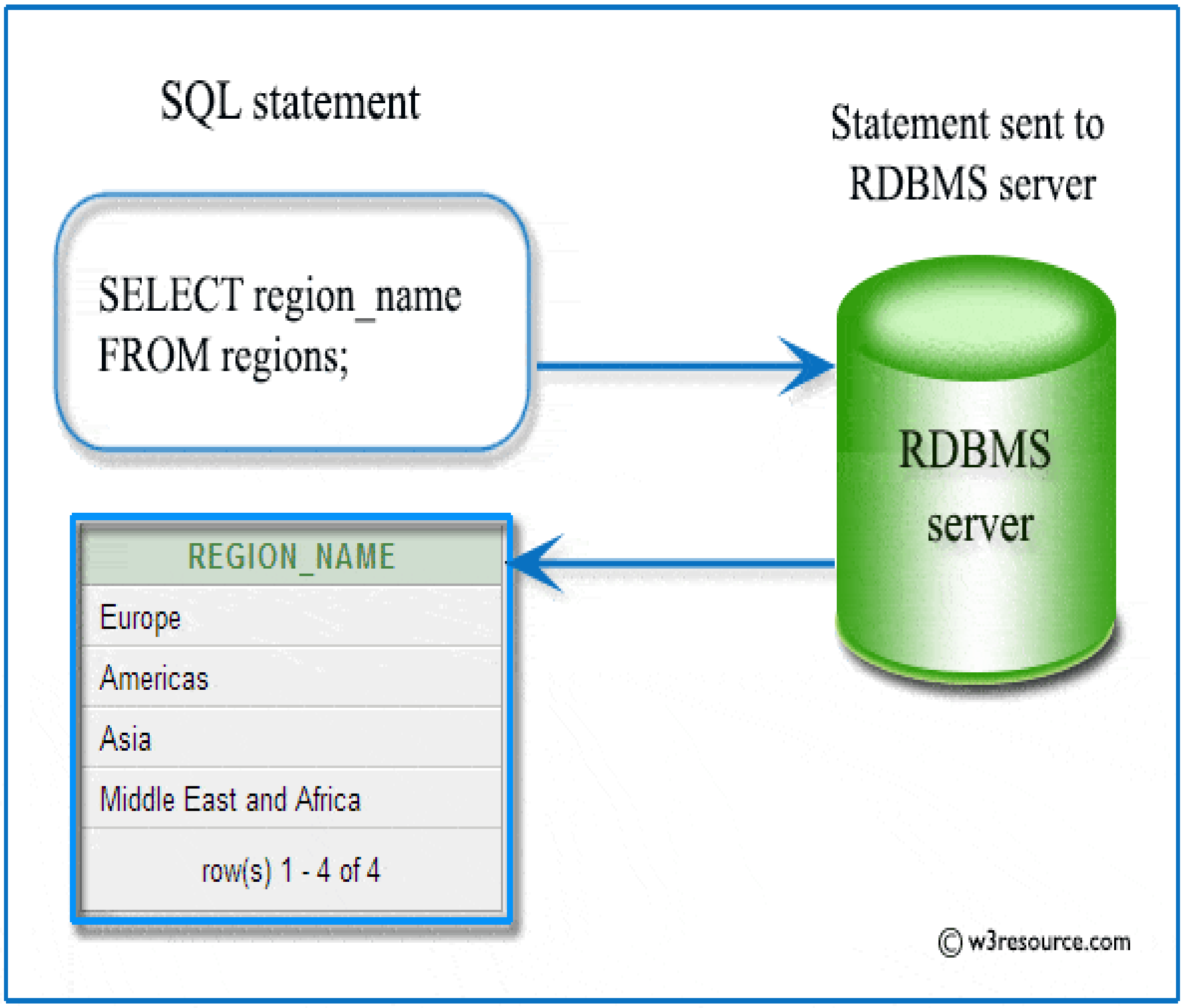
In 1979 Oracle introduced the first commercially available implementation of SQL. Later other players join in the race. Today, SQL is accepted as the standard RDBMS language.

Note: If you are not habituated with database management system your can learn from here**.**

What is SQL?

SQL stands for Structured Query Language and it is an ANSI (American National Standards Institute) standard computer language for accessing and manipulating database systems. It is used for managing data in relational database management system which stores data in the form of tables and relationship between data is also stored in the form of tables. SQL statements are used to retrieve and update data in a database.

SQL works with database programs like DB2, MySQL, PostgreSQL, Oracle, SQLite, SQL Server, Sybase, MS Access and much more. There are many different versions of the SQL language, but to be in compliance with the ANSI standard, they support the major keyword such as SELECT, UPDATE, DELETE, INSERT, WHERE, and others. The following picture shows the communicating with an RDBMS using SQL.



```
SELECT region_name
FROM regions;
```

REGION_NAME
Europe
Americas
Asia
Middle East and Africa
row(s) 1 - 4 of 4

History of SQL

Here is the year wise development history :

- 1970 E.F. Codd publishes Definition of Relational Model
- 1975 Initial version of SQL Implemented (D. Chamberlin)
- IBM experimental version: System R (1977) w/revised SQL
- IBM commercial versions: SQL/DS and DB2 (the early 1980s)
- Oracle introduces commercial version before IBM's SQL/DS
- INGRES 1981 & 85
- ShareBase 1982 & 86

- Data General (1984)
- Sybase (1986)
- by 1992 over 100 SQL products

SQL Standard Revisions

- SEQUEL/Original SQL - 1974
- SQL/86 - Ratification and acceptance of a formal SQL standard by ANSI (American National Standards Institute) and ISO (International Standards Organization).
- SQL/92 - Major revision (ISO 9075), Entry Level SQL-92 adopted as FIPS 127-2.
- SQL/99 - Added regular expression matching, recursive queries (e.g. transitive closure), triggers, support for procedural and control-of-flow statements, non-scalar types, and some object-oriented features (e.g. structured types).
- SQL/2003 - Introduced XML-related features (SQL/XML), Window functions, Auto generation.
- SQL/2006 - Lots of XML Support for XQuery, an XML-SQL interface standard.
- SQL/2008 - Adds INSTEAD OF triggers, TRUNCATE statement.

Constructs of SQL

Here is list of the key elements of SQL along with a brief description:

- Queries : Retrieves data against some criteria.
- Statements : Controls transactions, program flow, connections, sessions, or diagnostics.
- Clauses : Components of Queries and Statements.
- Expressions : Combination of symbols and operators and a key part of the SQL statements.
- Predicates : Specifies conditions.

Some Key terms of SQL 2003

To know the key terms of SQL 2003, you should know the statement classes of both SQL 92 AND SQL 2003, since both are used to refer SQL features and statements.
In SQL 92, SQL statements are grouped into following categories:

- **Data manipulation :** The Data Manipulation Language (DML) is the subset of SQL which is used to add, update and delete data.
- **Data definition :** The Data Definition Language (DDL) is used to manage table and index structure. CREATE, ALTER, RENAME, DROP and TRUNCATE statements are to name a few data definition elements.
- **Data control :** The Data Control Language (DCL) is used to set permissions to users and groups of users whether they can access and manipulate data.
- **Transaction :** A transaction contains a number of SQL statements. After the transaction begins, all of the SQL statements are executed and at the end of the transaction, permanent changes are made in the associated tables.
- **Procedure :** Using a stored procedure, a method is created which contains source code for performing repetitive tasks.

In SQL 2003 statements are grouped into seven categories which are called classes. See the following table :

Class	**Example**
SQL data statements	SELECT, INSERT, UPDATE, DELETE
SQL connection statements	CONNECT, DISCONNECT
SQL schema statements	ALTER, CREATE, DROP
SQL control statements	CALL, RETURN

SQL diagnostic statements	GET DIAGNOSTICS
SQL session statements	SET CONSTRAINT
SQL transaction statements	COMMIT, ROLLBACK

PL-SQL, TSQL and PL/pgSQL

- PL/SQL - Procedural Language/Structured Query Language (PL/SQL) is Oracle Corporation's procedural extension language for SQL and the Oracle relational database.
- TSQL - Transact-SQL (T-SQL) is Microsoft's and Sybase's proprietary extension to SQL.
- PL/pgSQL - Procedural Language/PostgreSQL(PL/pgSQL) is a procedural programming language supported by the PostgreSQL.

Database and Table Manipulation

Command	**Description**
CREATE DATABASE database_name	Create a database
DROP DATABASE database_name	Delete a database
CREATE TABLE "table_name" ("column_1" "column_1_data_type", "column_2" "column_2_data_type", ...)	Create a table in a database.

ALTER TABLE table_name ADD column_name column_datatype	Add columns in an existing table.
ALTER TABLE table_name DDROP column_name column_datatype	Delete columns in an existing table.
DROP TABLE table_name	Delete a table.

Data Types:

Data Type	**Description**
CHARACTER(n)	Character string, fixed length n.
CHARACTER VARYING(n) or VARCHAR(n)	Variable length character string, maximum length n.
BINARY(n)	Fixed-length binary string, maximum length n.
BOOLEAN	Stores truth values - either TRUE or FALSE.
BINARY VARYING(n) or VARBINARY(n)	Variable length binary string, maximum length n.
INTEGER(p)	Integer numerical, precision p.

SMALLINT	Integer numerical precision 5.
INTEGER	Integer numerical, precision 10.
BIGINT	Integer numerical, precision 19.
DECIMAL(p, s)	Exact numerical, precision p, scale s.
NUMERIC(p, s)	Exact numerical, precision p, scale s. (Same as DECIMAL).
FLOAT(p)	Approximate numerical, mantissa precision p.
REAL	Approximate numerical mantissa precision 7.
FLOAT	Approximate numerical mantissa precision 16.
DOUBLE PRECISION	Approximate numerical mantissa precision 16.
DATE TIME TIMESTAMP	Composed of a number of integer fields, representing an absolute point in time, depending on sub-type.
INTERVAL	Composed of a number of integer fields, representing a period of

	time, depending on the type of interval.
COLLECTION (ARRAY, MULTISET)	ARRAY(offered in SQL99) is a set-length and ordered the collection of elements.
XML	Stores XML data. It can be used wherever a SQL data type is allowed, such as a column of a table.

Index Manipulation:

Command	**Description**
CREATE INDEX index_name ON table_name (column_name_1, column_name_2, ...)	Create a simple index.
CREATE UNIQUE INDEX index_name ON table_name (column_name_1, column_name_2, ...)	Create a unique index.
DROP INDEX table_name.index_name	Drop a index.

SQL Operators:

Operators	**Description**
SQL Arithmetic	Arithmetic operators are addition(+), subtraction(-), multiplication(*) and

Operator	division(/). The + and - operators can also be used in date arithmetic.
SQL Comparison Operator	A comparison (or relational) operator is a mathematical symbol which is used to compare two values.
SQL Assignment operator	In SQL the assignment operator (=) assigns a value to a variable or of a column or field of a table.
SQL Bitwise Operator	The bitwise operators are & (Bitwise AND), | (Bitwise OR) and ^ (Bitwise Exclusive OR or XOR). The valid datatypes for bitwise operators are BINARY, BIT, INT, SMALLINT, TINYINT, and VARBINARY.
SQL Logical Operator	The Logical operators are those that are true or false. The logical operators are AND , OR, NOT, IN, BETWEEN, ANY, ALL, SOME, EXISTS, and LIKE.
SQL Unary Operator	The SQL Unary operators perform such an operation which contain only one expression of any of the datatypes in the numeric datatype category.

Insert, Update and Delete:

Command	**Description**

INSERT INTO table_name VALUES (value_1, value_2,....)
INSERT INTO table_name (column1, column2,...) VALUES (value_1, value_2,....)
	Insert new rows into a table.

UPDATE table_name SET column_name_1 = new_value_1, column_name_2 = new_value_2 WHERE column_name = some_value	Update one or several columns in rows.
DELETE FROM table_name WHERE column_name = some_value	Delete rows in a table.

Select:

Command	**Description**
SELECT column_name(s) FROM table_name	Select data from a table.
SELECT * FROM table_name	Select all data from a table.
SELECT DISTINCT column_name(s) FROM table_name	Select only distinct (different) data from a table.
SELECT column_name(s) FROM table_name WHERE column operator value AND column operator value OR column operator value AND (... OR ...) ...	Select only certain data from a table.
SELECT column_name(s) FROM table_name WHERE column_name IN (value1, value2, ...)	The IN operator may be used if you know the exact value you want to return for at least one of the columns.
SELECT column_name(s) FROM table_name ORDER BY row_1, row_2	Select data from a table with sort the rows.

DESC, row_3 ASC, ...	
SELECT column_1, ..., SUM(group_column_name) FROM table_name GROUP BY group_column_name	The GROUP BY clause is used with the SELECT statement to make a group of rows based on the values of a specific column or expression. The SQL AGGREGATE function can be used to get summary information for every group and these are applied to individual group.
SELECT column_name(s) INTO new_table_name FROM source_table_name WHERE query	Select data from table(S) and insert it into another table.
SELECT column_name(s) IN external_database_name FROM source_table_name WHERE query	Select data from table(S) and insert it in another database.

Functions:

SQL functions	**Description**
Aggregate Function	This function can produce a single value for an entire group or table. Some Aggregate functions are - • SQL Count function • SQL Sum function • SQL Avg function • SQL Max function • SQL Min function

Arithmetic Function	A mathematical function executes a mathematical operation usually based on input values that are provided as arguments, and return a numeric value as the result of the operation. Some Arithmetic functions are - • abs() • ceil() • floor() • exp() • ln() • mod() • power() • sqrt()
Character Function	A character or string function is a function which takes one or more characters or numbers as parameters and returns a character value. Some Character functions are - • lower() • upper() • trim() • translate()

Joins:

Name	**Description**
SQL EQUI JOIN	The SQL EQUI JOIN is a simple SQL join uses the equal sign(=) as the comparison operator for the condition. It has two types - SQL Outer join and SQL Inner join. SQL INNER JOIN returns all rows from tables where the key record of one table is equal to the key records of another table. SQL OUTER JOIN returns all rows from one table and only those rows from the

	secondary table where the joined condition is satisfying i.e. the columns are equal in both tables.
SQL NON EQUI JOIN	The SQL NON EQUI JOIN is a join uses comparison operator other than the equal sign like >, <, >=, <= with the condition.

Union:

Command	**Description**
SQL_Statement_1 UNION SQL_Statement_2	Select all different values from SQL_Statement_1 and SQL_Statement_2
SQL_Statement_1 UNION ALL SQL_Statement_2	Select all values from SQL_Statement_1 and SQL_Statement_2

View:

Command	**Description**
CREATE VIEW view_name AS SELECT column_name(s) FROM table_name WHERE condition	Create a virtual table based on the result-set of a SELECT statement.

Tutorial objectives

SQL tutorial of w3resource is a comprehensive tutorial to learn SQL. We have followed SQL:2003 standard of ANSI. There are hundreds of examples given in this tutorial. Output are shown with Oracle 10G/MySQL. Often outputs are followed by a pictorial presentation and explanation for better understanding. You will hardly find a vendor neutral SQL tutorial covering SQL in such great detail. Following is a list of the features we have included in our tutorials :

- A simple but thorough description.
- SQL Syntax.
- Description of the Parameters used in the SQL command.
- Sample table with data.
- SQL command.
- Explanation of the SQL command.
- The output of the SQL command.
- Model database.
- Online practice.

Summary

- SQL stands for Structured Query Language.
- SQL is easy to learn.
- SQL is an ANSI standard computer language.
- SQL allows us to access a database.
- SQL use to access and manipulate data in various databases like Oracle, Sybase, Microsoft SQL Server, DB2, Access, MySQL, PostgreSQL and other database systems.

- SQL execute queries against a database.
- SQL can insert new records into a database.
- SQL can update records in a database.
- SQL can delete records from a database.

Practice SQL Exercises

- SQL Exercises, Practice, Solution
- SQL Retrieve data from tables [33 Exercises]
- SQL Boolean and Relational operators [12 Exercises]
- SQL Wildcard and Special operators [22 Exercises]
- SQL Aggregate Functions [25 Exercises]
- SQL Formatting query output [10 Exercises]
- SQL Quering on Multiple Tables [7 Exercises]
- FILTERING and SORTING on HR Database [38 Exercises]
- SQL JOINS
    - SQL JOINS [29 Exercises]
    - SQL JOINS on HR Database [27 Exercises]
- SQL SUBQUERIES
    - SQL SUBQUERIES [39 Exercises]
    - SQL SUBQUERIES on HR Database [55 Exercises]
- SQL Union[9 Exercises]
- SQL View[16 Exercises]
- SQL User Account Management [16 Exercise]
- Movie Database
    - BASIC queries on movie Database [10 Exercises]
    - SUBQUERIES on movie Database [16 Exercises]

  - JOINS on movie Database [24 Exercises]
- Soccer Database
  - Introduction
  - BASIC queries on soccer Database [29 Exercises]
  - SUBQUERIES on soccer Database [33 Exercises]
  - JOINS queries on soccer Database [61 Exercises]
- Hospital Database
  - Introduction
  - BASIC, SUBQUERIES, and JOINS [39 Exercises]
- Employee Database
  - BASIC queries on employee Database [115 Exercises]
  - SUBQUERIES on employee Database [77 Exercises]
- More to come!

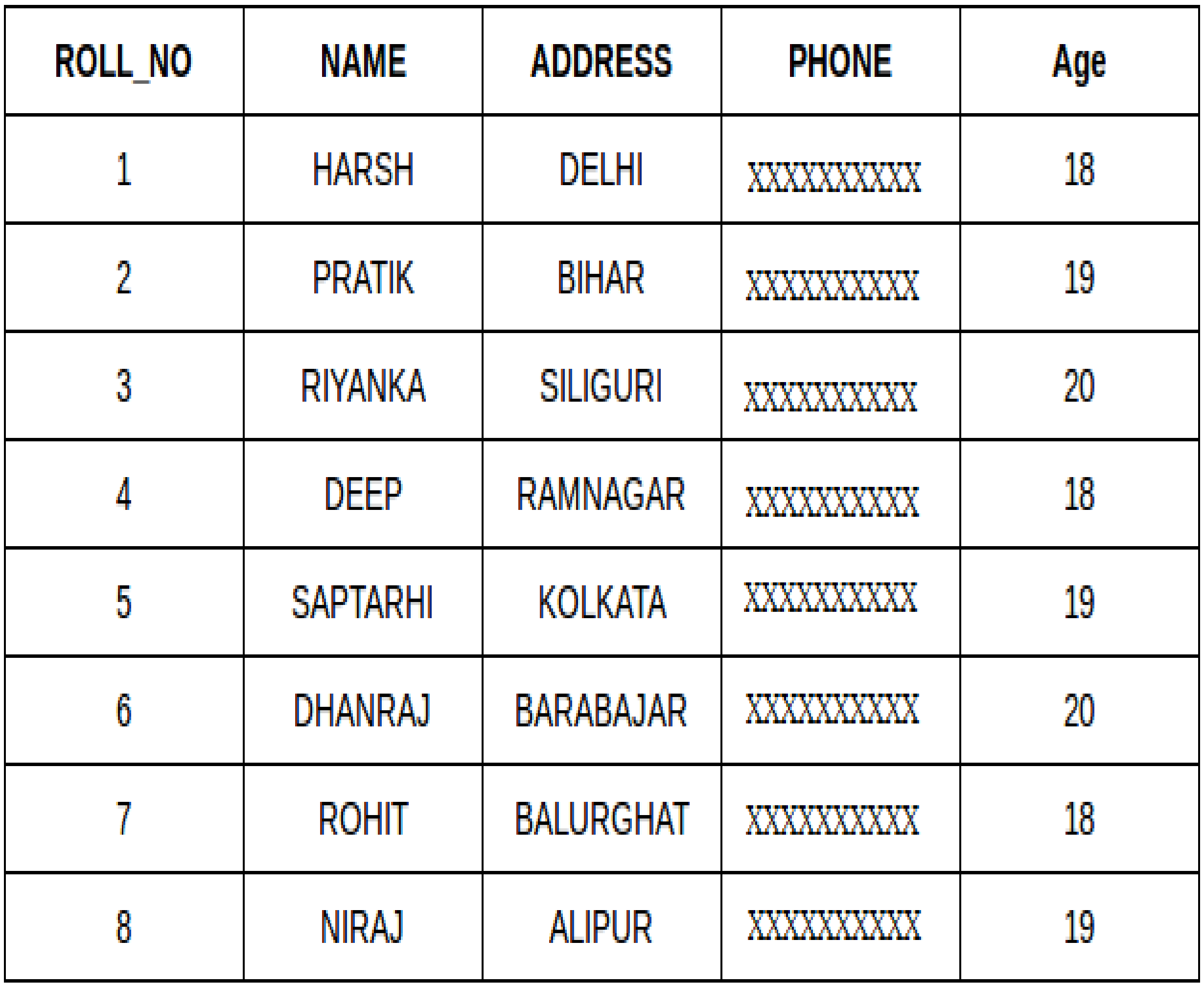
SQL Join

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

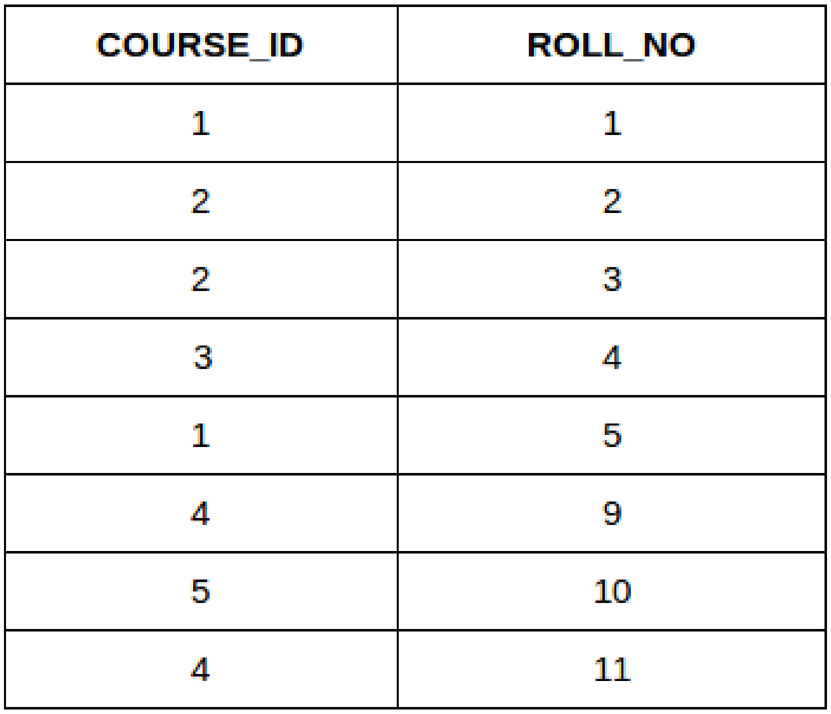
Consider the two tables below:

Student



ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

StudentCourse



COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

The simplest Join is INNER JOIN.

1. **INNER JOIN:** The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.

   Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

INNER JOIN table2

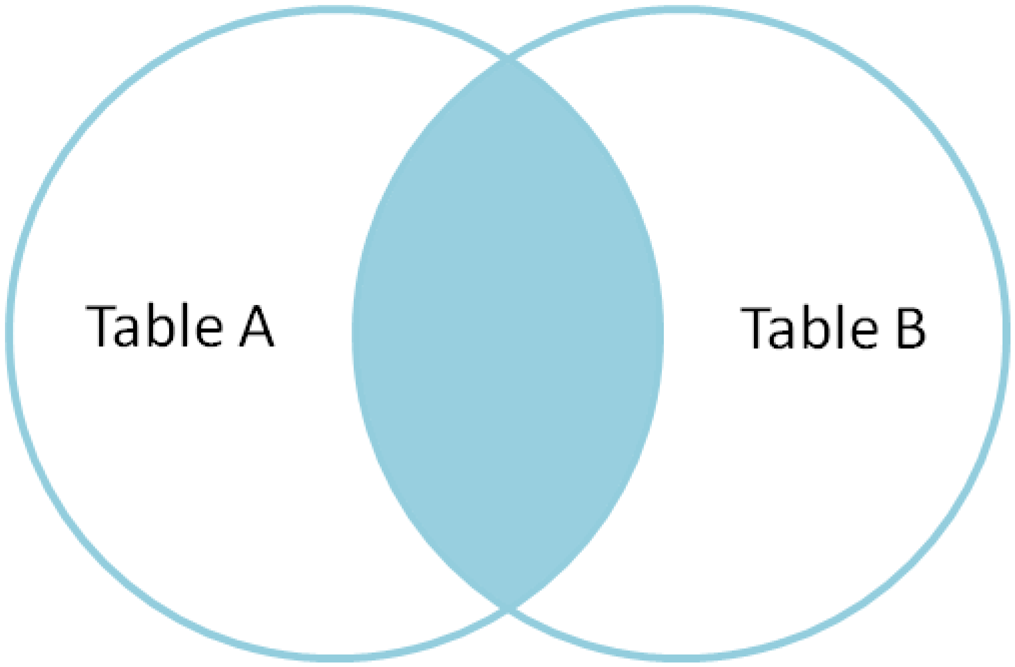
ON table1.matching_column = table2.matching_column;

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Note: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.



Example Queries(INNER JOIN)

- This query will show the names and age of students enrolled in different courses.

```
• SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM
  Student
• INNER JOIN StudentCourse
• ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

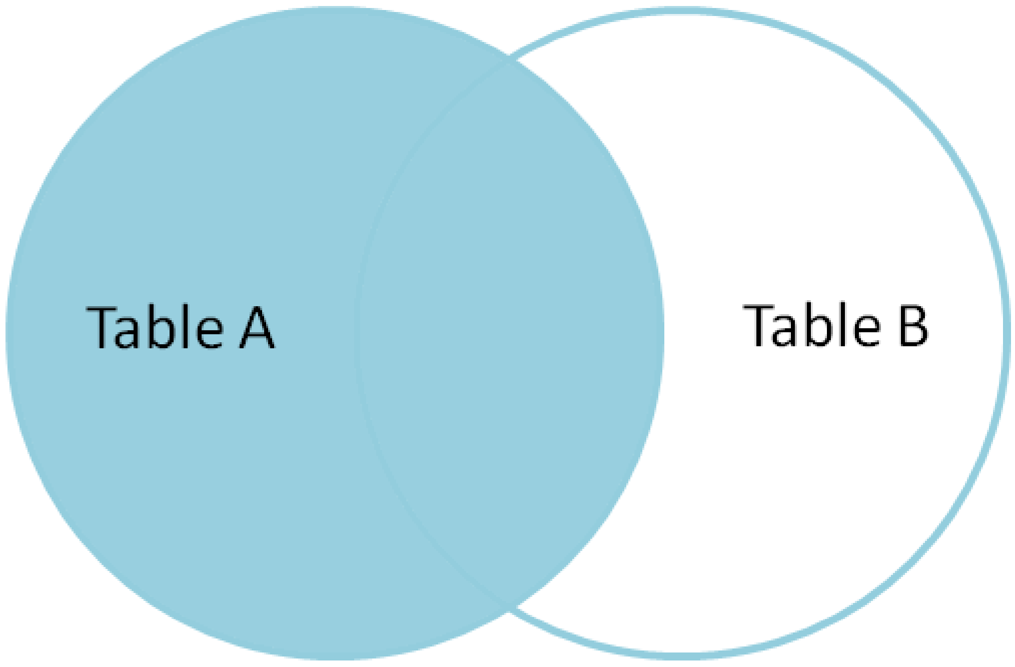
Output:

2. **LEFT JOIN**: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join. The rows for which there is no matching row on right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN.**Syntax:**

```
3. SELECT table1.column1,table1.column2,table2.column1,....
4. FROM table1
5. LEFT JOIN table2
```

```
6.  ON table1.matching_column = table2.matching_column;
7.
8.
9.  table1: First table.
10. table2: Second table
11. matching_column: Column common to both the tables.
```

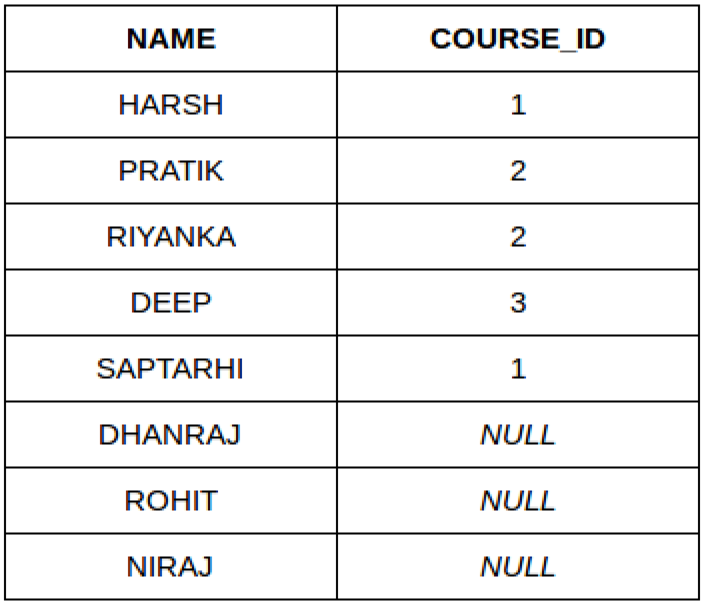
Note: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are same.



Example Queries(LEFT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

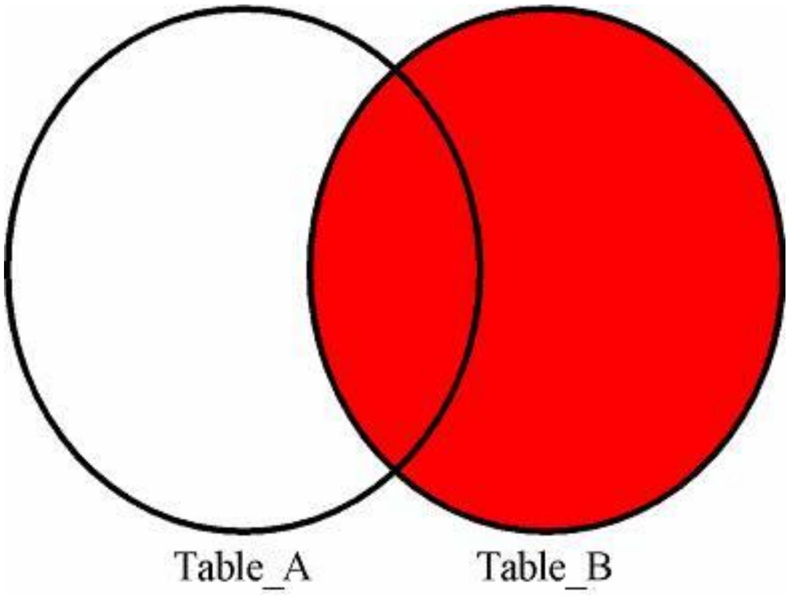


NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

12. **RIGHT JOIN**: RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.**Syntax:**

```
13. SELECT table1.column1,table1.column2,table2.column1,....
14. FROM table1
15. RIGHT JOIN table2
16. ON table1.matching_column = table2.matching_column;
17.
18.
19. table1: First table.
20. table2: Second table
21. matching_column: Column common to both the tables.
```

Note: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are same.



Example Queries(RIGHT JOIN):

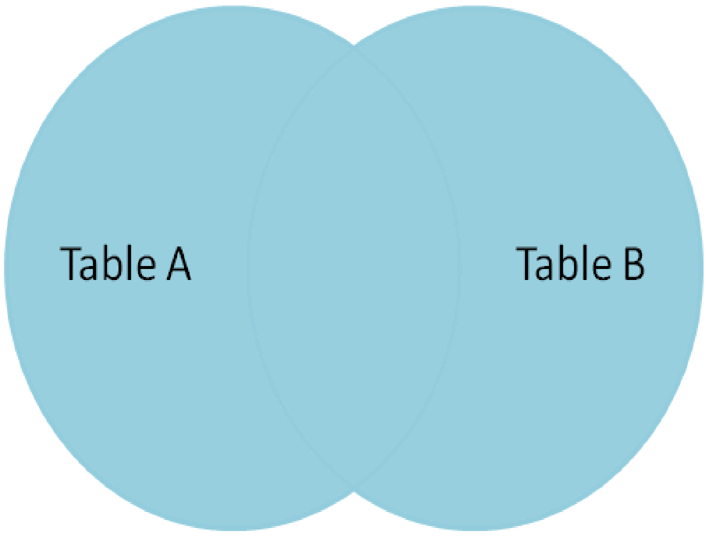
```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

22. **FULL JOIN:** FULL JOIN creates the result-set by combining result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain NULL values.**Syntax:**

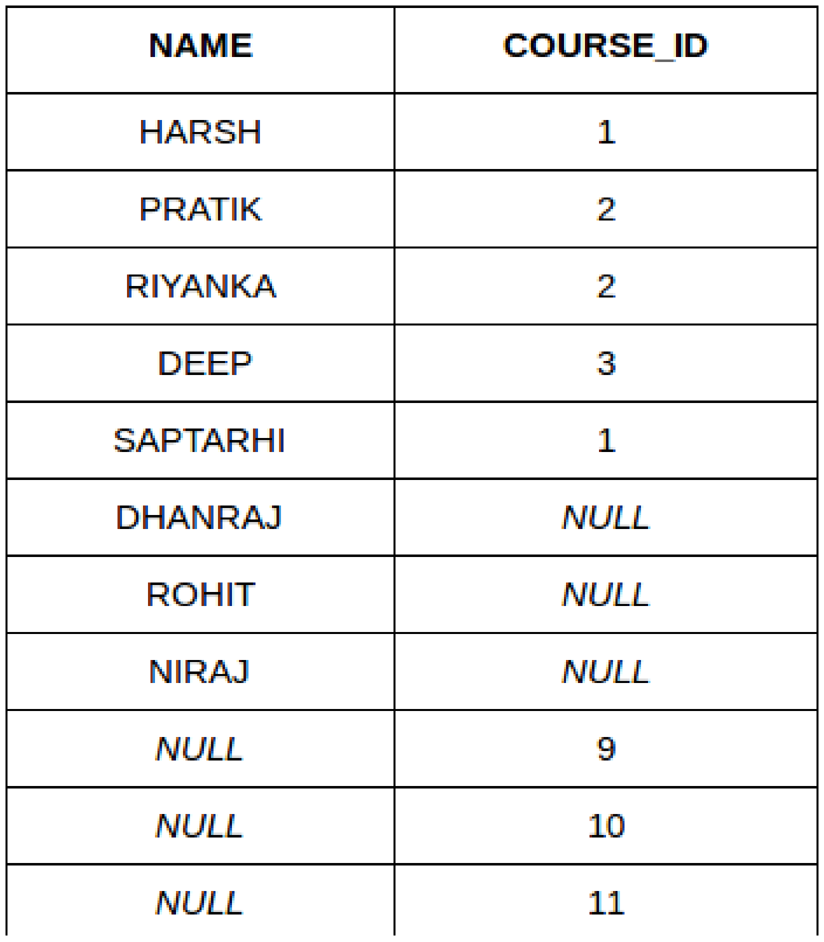
```
23. SELECT table1.column1,table1.column2,table2.column1,....
24. FROM table1
25. FULL JOIN table2
26. ON table1.matching_column = table2.matching_column;
27.
28.
29. table1: First table.
30. table2: Second table
31. matching_column: Column common to both the tables.
```



Example Queries(FULL JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:



NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*
NULL	9
NULL	10
NULL	11

Multiple Table Queries

It’s sometimes difficult to know which SQL syntax to use when combining data that spans multiple tables. I’ll discuss some of the more frequently used methods for consolidating queries on multiple tables into a single statement.

SQL syntax

If you need a refresher on SQL syntax, read these articles:
"SQL Basics I: Data queries" covers database terminology and the four basic query types.
"SQL basics: SELECT statement options" covers the SELECT statement in detail and explains aggregate functions.

The sample queries in this article adhere to the SQL92 ISO standard. Not all database manufacturers follow this standard, and many have made enhancements that can yield unexpected results. If you're uncertain about support for these concepts in your database, please refer to your manufacturer's documentation.

SELECT

A simple SELECT statement is the most basic way to query multiple tables. You can call more than one table in the FROM clause to combine results from multiple tables. Here's an example of how this works:
SELECT table1.column1, table2.column2 FROM table1, table2 WHERE table1.column1 = table2.column1;

In this example, I used dot notation (table1.column1) to specify which table the column came from. If the column in question only appears in one of the referenced tables, you don't need to include the fully qualified name, but it may be useful to do so for readability.

Tables are separated in the FROM clause by commas. You can include as many tables as needed, although some databases have a limit to what they can efficiently handle before introducing a formal JOIN statement, which is described below.

This syntax is, in effect, a simple INNER JOIN. Some databases treat it exactly the same as an explicit JOIN. The WHERE clause tells the database which fields to correlate, and it returns results as if the tables listed were combined into a single table based on the provided conditions. It's worth noting that your conditions for comparison don't have to be the same columns you return as your result set. In the example above, table1.column1 and table2.column1 are used to combine the tables, but table2.column2 is returned.

You can extend this functionality to more than two tables using AND keywords in the WHERE clause. You can also use such a combination of tables to restrict your results without actually returning columns from every table. In the example below, table3 is matched up with table1, but I haven't returned anything from table3 for display. I've merely checked to make sure the relevant column from table1 exists in table3. Note that table3 needs to be referenced in the FROM clause for this example.
SELECT table1.column1, table2.column2 FROM table1, table2, table3 WHERE table1.column1 = table2.column1 AND table1.column1 = table3.column1;

Be warned, however, that this method of querying multiple tables is effectively an implied JOIN. Your database may handle things differently, depending on the optimization engine it uses. Also, neglecting to define the nature of the correlation with a WHERE clause can give you undesirable results, such as returning the rogue field in a column associated with every possible result from the rest of the query, as in a CROSS JOIN.

If you're comfortable with how your database handles this type of statement, and you're combining two or just a few tables, a simple SELECT statement will meet your needs.

JOIN

JOIN works in the same way as the SELECT statement above—it returns a result set with columns from different tables. The advantage of using an explicit JOIN over an implied one is greater control over your result set, and possibly improved performance when many tables are involved.

There are several types of JOIN—LEFT, RIGHT, and FULL OUTER; INNER; and CROSS. The type you use is determined by the results you want to see. For example, using a LEFT OUTER JOIN will return all relevant rows from the first table listed, while potentially dropping rows from the second table listed if they don't have information that correlates in the first table.

This differs from an INNER JOIN or an implied JOIN. An INNER JOIN will only return rows for which there is data in both tables.

Use the following JOIN statement for the first SELECT query above:

```
SELECT table1.column1, table2.column2 FROM table1 INNER JOIN table2
ON table1.column1 = table2.column1;
```

Subqueries

Subqueries, or subselect statements, are a way to use a result set as a resource in a query. These are often used to limit or refine results rather than run multiple queries or manipulate the data in your application. With a subquery, you can reference tables to determine inclusion of data or, in some cases, return a column that is the result of a subselect.

The following example uses two tables. One table actually contains the data I'm interested in returning, while the other gives a comparison point to determine what data is actually interesting.

```
SELECT column1 FROM table1 WHERE EXISTS ( SELECT column1 FROM table2
WHERE table1.column1 = table2.column1 );
```

One important factor about subqueries is performance. Convenience comes at a price and, depending on the size, number, and complexity of tables and the statements you use, you may want to allow your application to handle processing. Each query is processed separately in full before being used as a resource for your primary query. If possible, creative use of JOIN statements may provide the same information with less lag time.

JOIN statements and subqueries

For a more detailed explanation of JOINS and concepts that can be used with them, read the articles "Basic and complex SQL joins made easy" and "Master joins with these concepts." For more information about subqueries, read "Use SQL subselects to consolidate queries."

UNION

The UNION statement is another way to return information from multiple tables with a single query. The UNION statement allows you to perform queries against several tables and return the results in a consolidated set, as in the following example.
SELECT column1, column2, column3 FROM table1 UNION SELECT column1, column2, column3 FROM table2;

This will return a result set with three columns containing data from both queries. By default, the UNION statement will omit duplicates between the tables unless the UNION ALL keyword is used. UNION is helpful when the returned columns from the different tables don't have columns or data that can be compared and joined, or when it prevents running multiple queries and appending the results in your application code.

If your column names don't match when you use the UNION statement, use aliases to give your results meaningful headers:
SELECT column1, column2 as Two, column3 as Three FROM table1 UNION SELECT column1, column4 as Two, column5 as Three FROM table2;

As with subqueries, UNION statements can create a heavy load on your database server, but for occasional use they can save a lot of time.

Multiple options
When it comes to database queries, there are usually many ways to approach the same problem. These are some of the more frequently used methods for consolidating queries on multiple tables into a single statement. While some of these options may affect performance, practice will help you know when it's appropriate to use each type of query.

Build-in functions

The Python built-in functions are defined as the functions whose functionality is pre-defined in Python. The python interpreter has several functions that are always present for use. These functions are known as Built-in Functions. There are several built-in functions in Python which are listed below:

Python abs() Function

The python **abs()** function is used to return the absolute value of a number. It takes only one argument, a number whose absolute value is to be returned. The argument can be an integer and floating-point number. If the argument is a complex number, then, abs() returns its magnitude.

Python abs() Function Example

```
#  integer number
integer = -20
print('Absolute value of -40 is:', abs(integer))

#  floating number
floating = -20.83
print('Absolute value of -40.83 is:', abs(floating))
```

Output:

```
Absolute value of -20 is: 20
Absolute value of -20.83 is: 20.83
```

Python all() Function

The python **all()** function accepts an iterable object (such as list, dictionary, etc.). It returns true if all items in passed iterable are true. Otherwise, it returns False. If the iterable object is empty, the all() function returns True.

Python all() Function Example

```
# all values true
k = [1, 3, 4, 6]
print(all(k))

# all values false
k = [0, False]
print(all(k))

```

```
# one false value
k = [1, 3, 7, 0]
print(all(k))

# one true value
k = [0, False, 5]
print(all(k))

# empty iterable
k = []
print(all(k))
```

Output:

```
True
False
False
False
True
```

Python bin() Function

The python **bin()** function is used to return the binary representation of a specified integer. A result always starts with the prefix 0b.

Python bin() Function Example

```
x =  10
y =  bin(x)
print (y)
```

Output:

```
0b1010
```

Python bool()

The python **bool()** converts a value to boolean(True or False) using the standard truth testing procedure.

Python bool() Example

```
test1 = []
print(test1,'is',bool(test1))
```

```
test1 = [0]
print(test1,'is',bool(test1))
test1 = 0.0
print(test1,'is',bool(test1))
test1 = None
print(test1,'is',bool(test1))
test1 = True
print(test1,'is',bool(test1))
test1 = 'Easy string'
print(test1,'is',bool(test1))
```

Output:

```
[] is False
[0] is True
0.0 is False
None is False
True is True
Easy string is True
```

Python bytes()

The python **bytes()** in Python is used for returning a **bytes** object. It is an immutable version of the bytearray() function.

It can create empty bytes object of the specified size.

Python bytes() Example

```
string = "Hello World."
array = bytes(string, 'utf-8')
print(array)
```

Output:

```
b ' Hello World.'
```

Python callable() Function

A python **callable()** function in Python is something that can be called. This built-in function checks and returns true if the object passed appears to be callable, otherwise false.

Python callable() Function Example

```
x = 8
print(callable(x))
```

Output:

```
False
```

Python compile() Function

The python **compile()** function takes source code as input and returns a code object which can later be executed by exec() function.

Python compile() Function Example

```
# compile string source to code
code_str = 'x=5\ny=10\nprint("sum =",x+y)'
code = compile(code_str, 'sum.py', 'exec')
print(type(code))
exec(code)
exec(x)
```

Output:

```
<class 'code'>
sum = 15
```

Python exec() Function

The python **exec()** function is used for the dynamic execution of Python program which can either be a string or object code and it accepts large blocks of code, unlike the eval() function which only accepts a single expression.

Python exec() Function Example

```
x = 8
exec('print(x==8)')
exec('print(x+4)')
```

Output:

```
True
12
```

Python sum() Function

As the name says, python **sum()** function is used to get the sum of numbers of an iterable, i.e., list.

Python sum() Function Example

```
s = sum([1, 2,4 ])
print(s)

s = sum([1, 2, 4], 10)
print(s)
```

Output:

```
7
17
```

Python any() Function

The python **any()** function returns true if any item in an iterable is true. Otherwise, it returns False.

Python any() Function Example

```
l = [4, 3, 2, 0]
print(any(l))

l = [0, False]
print(any(l))

l = [0, False, 5]
print(any(l))

l = []
print(any(l))
```

Output:

```
True
False
True
False
```

Python ascii() Function

The python **ascii()** function returns a string containing a printable representation of an object and escapes the non-ASCII characters in the string using \x, \u or \U escapes.

Python ascii() Function Example

```
normalText = 'Python is interesting'
print(ascii(normalText))

otherText = 'Pythön is interesting'
print(ascii(otherText))

print('Pyth\xf6n is interesting')
```

Output:

```
'Python is interesting'
'Pyth\xf6n is interesting'
Pythön is interesting
```

Python bytearray()

The python **bytearray()** returns a bytearray object and can convert objects into bytearray objects, or create an empty bytearray object of the specified size.

Python bytearray() Example

```
string = "Python is a programming language."

# string with encoding 'utf-8'
arr = bytearray(string, 'utf-8')
print(arr)
```

Output:

```
bytearray(b'Python is a programming language.')
```

Python eval() Function

The python **eval()** function parses the expression passed to it and runs python expression(code) within the program.

Python eval() Function Example

```python
x = 8
print(eval('x + 1'))
```

Output:

```
9
```

Python float()

The python **float()** function returns a floating-point number from a number or string.

Python float() Example

```python
# for integers
print(float(9))

# for floats
print(float(8.19))

# for string floats
print(float("-24.27"))

# for string floats with whitespaces
print(float("     -17.19\n"))

# string float error
print(float("xyz"))
```

Output:

```
9.0
8.19
-24.27
-17.19
ValueError: could not convert string to float: 'xyz'
```

Python format() Function

The python **format()** function returns a formatted representation of the given value.

Python format() Function Example

```python
# d, f and b are a type

# integer
print(format(123, "d"))

# float arguments
print(format(123.4567898, "f"))

# binary format
print(format(12, "b"))
```

Output:

```
123
123.456790
1100
```

Python frozenset()

The python **frozenset()** function returns an immutable frozenset object initialized with elements from the given iterable.

Python frozenset() Example

```
# tuple of letters
letters = ('m', 'r', 'o', 't', 's')

fSet = frozenset(letters)
print('Frozen set is:', fSet)
print('Empty frozen set is:', frozenset())
```

Output:

```
Frozen set is: frozenset({'o', 'm', 's', 'r', 't'})
Empty frozen set is: frozenset()
```

Python getattr() Function

The python **getattr()** function returns the value of a named attribute of an object. If it is not found, it returns the default value.

Python getattr() Function Example

```
class Details:
    age = 22
    name = "Phill"
```

```python

details = Details()
print('The age is:', getattr(details, "age"))
print('The age is:', details.age)
```

Output:

```
The age is: 22
The age is: 22
```

Python globals() Function

The python **globals()** function returns the dictionary of the current global symbol table.

A **Symbol table** is defined as a data structure which contains all the necessary information about the program. It includes variable names, methods, classes, etc.

Python globals() Function Example

```python
age = 22

globals()['age'] = 22
print('The age is:', age)
```

Output:

```
The age is: 22
```

Python hasattr() Function

The python **any()** function returns true if any item in an iterable is true, otherwise it returns False.

Python hasattr() Function Example

```python
l = [4, 3, 2, 0]
print(any(l))

l = [0, False]
print(any(l))

l = [0, False, 5]
print(any(l))

```

```
l = []
print(any(l))
```

Output:

```
True
False
True
False
```

Python iter() Function

The python **iter()** function is used to return an iterator object. It creates an object which can be iterated one element at a time.

Python iter() Function Example

```
# list of numbers
list = [1,2,3,4,5]

listIter = iter(list)

# prints '1'
print(next(listIter))

# prints '2'
print(next(listIter))

# prints '3'
print(next(listIter))

# prints '4'
print(next(listIter))

# prints '5'
print(next(listIter))
```

Output:

```
1
2
3
4
5
```

Python len() Function

The python **len()** function is used to return the length (the number of items) of an object.

Python len() Function Example

```
strA = 'Python'
print(len(strA))
```

Output:

```
6
```

Python list()

The python **list()** creates a list in python.

Python list() Example

```
# empty list
print(list())

# string
String = 'abcde'
print(list(String))

# tuple
Tuple = (1,2,3,4,5)
print(list(Tuple))
# list
List = [1,2,3,4,5]
print(list(List))
```

Output:

```
[]
['a', 'b', 'c', 'd', 'e']
[1,2,3,4,5]
[1,2,3,4,5]
```

Python locals() Function

The python **locals()** method updates and returns the dictionary of the current local symbol table.

A **Symbol table** is defined as a data structure which contains all the necessary information about the program. It includes variable names, methods, classes, etc.

Python locals() Function Example

```
def localsAbsent():
    return locals()

def localsPresent():
    present = True
    return locals()

print('localsNotPresent:', localsAbsent())
print('localsPresent:', localsPresent())
```

Output:

```
localsAbsent: {}
localsPresent: {'present': True}
```

Python map() Function

The python **map()** function is used to return a list of results after applying a given function to each item of an iterable(list, tuple etc.).

Python map() Function Example

```
def calculateAddition(n):
  return n+n

numbers = (1, 2, 3, 4)
result = map(calculateAddition, numbers)
print(result)

# converting map object to set
numbersAddition = set(result)
print(numbersAddition)
```

Output:

```
<map object at 0x7fb04a6bec18>
{8, 2, 4, 6}
```

Python memoryview() Function

The python **memoryview()** function returns a memoryview object of the given argument.

Python memoryview () Function Example

```
#A random bytearray
randomByteArray = bytearray('ABC', 'utf-8')

mv = memoryview(randomByteArray)

# access the memory view's zeroth index
print(mv[0])

# It create byte from memory view
print(bytes(mv[0:2]))

# It create list from memory view
print(list(mv[0:3]))
```

Output:

```
65
b'AB'
[65, 66, 67]
```

Python object()

The python **object()** returns an empty object. It is a base for all the classes and holds the built-in properties and methods which are default for all the classes.

Python object() Example

```
python = object()

print(type(python))
print(dir(python))
```

Output:

```
<class 'object'>
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__']
```

Python open() Function

The python **open()** function opens the file and returns a corresponding file object.

Python open() Function Example

```
# opens python.text file of the current directory
f = open("python.txt")
# specifying full path
f = open("C:/Python33/README.txt")
```

Output:

```
Since the mode is omitted, the file is opened in 'r' mode; opens for reading.
```

Python chr() Function

Python **chr()** function is used to get a string representing a character which points to a Unicode code integer. For example, chr(97) returns the string 'a'. This function takes an integer argument and throws an error if it exceeds the specified range. The standard range of the argument is from 0 to 1,114,111.

Python chr() Function Example

```
# Calling function
result = chr(102) # It returns string representation of a char
result2 = chr(112)
# Displaying result
print(result)
print(result2)
# Verify, is it string type?
print("is it string type:", type(result) is str)
```

Output:

```
ValueError: chr() arg not in range(0x110000)
```

Python complex()

Python **complex()** function is used to convert numbers or string into a complex number. This method takes two optional parameters and returns a complex number. The first parameter is called a real and second as imaginary parts.

Python complex() Example

```
# Python complex() function example
# Calling function
a = complex(1) # Passing single parameter
b = complex(1,2) # Passing both parameters
# Displaying result
print(a)
print(b)
```

Output:

```
(1.5+0j)
(1.5+2.2j)
```

Python delattr() Function

Python **delattr()** function is used to delete an attribute from a class. It takes two parameters, first is an object of the class and second is an attribute which we want to delete. After deleting the attribute, it no longer available in the class and throws an error if try to call it using the class object.

Python delattr() Function Example

```
class Student:
    id = 101
    name = "Pranshu"
    email = "pranshu@abc.com"
# Declaring function
    def getinfo(self):
        print(self.id, self.name, self.email)
s = Student()
s.getinfo()
delattr(Student,'course') # Removing attribute which is not available
s.getinfo() # error: throws an error
```

Output:

```
101 Pranshu pranshu@abc.com
AttributeError: course
```

Python dir() Function

Python **dir()** function returns the list of names in the current local scope. If the object on which method is called has a method named __dir__(), this method will be called and must return the list of attributes. It takes a single object type argument.

Python dir() Function Example

```
# Calling function
att = dir()
# Displaying result
print(att)
```

Output:

```
['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
'__name__', '__package__', '__spec__']
```

Python divmod() Function

Python **divmod()** function is used to get remainder and quotient of two numbers. This function takes two numeric arguments and returns a tuple. Both arguments are required and numeric

Python divmod() Function Example

```
# Python divmod() function example
# Calling function
result = divmod(10,2)
# Displaying result
print(result)
```

Output:

```
(5, 0)
```

Python enumerate() Function

Python **enumerate()** function returns an enumerated object. It takes two parameters, first is a sequence of elements and the second is the start index of the sequence. We can get the elements in sequence either through a loop or next() method.

Python enumerate() Function Example

```
# Calling function
result = enumerate([1,2,3])
# Displaying result
print(result)
print(list(result))
```

Output:

```
<enumerate object at 0x7ff641093d80>
[(0, 1), (1, 2), (2, 3)]
```

Python dict()

Python **dict()** function is a constructor which creates a dictionary. Python dictionary provides three different constructors to create a dictionary:

- If no argument is passed, it creates an empty dictionary.
- If a positional argument is given, a dictionary is created with the same key-value pairs. Otherwise, pass an iterable object.
- If keyword arguments are given, the keyword arguments and their values are added to the dictionary created from the positional argument.

Python dict() Example

```
# Calling function
result = dict() # returns an empty dictionary
result2 = dict(a=1,b=2)
# Displaying result
print(result)
print(result2)
```

Output:

```
{}
{'a': 1, 'b': 2}
```

Python filter() Function

Python **filter()** function is used to get filtered elements. This function takes two arguments, first is a function and the second is iterable. The filter function returns a sequence of those elements of iterable object for which function returns **true value**.

The first argument can be **none**, if the function is not available and returns only elements that are **true**.

Python filter() Function Example

```
# Python filter() function example
def filterdata(x):
    if x>5:
        return x
# Calling function
result = filter(filterdata,(1,2,6))
# Displaying result
print(list(result))
```

Output:

```
[6]
```

Python hash() Function

Python **hash()** function is used to get the hash value of an object. Python calculates the hash value by using the hash algorithm. The hash values are integers and used to compare dictionary keys during a dictionary lookup. We can hash only the types which are given below:

Hashable types: * bool * int * long * float * string * Unicode * tuple * code object.

Python hash() Function Example

```
# Calling function
result = hash(21) # integer value
result2 = hash(22.2) # decimal value
# Displaying result
print(result)
print(result2)
```

Output:

```
21
461168601842737174
```

Python help() Function

Python **help()** function is used to get help related to the object passed during the call. It takes an optional parameter and returns help information. If no argument is given, it shows the Python help console. It internally calls python's help function.

Python help() Function Example

1. # Calling function
2. info = help() # No argument
3. # Displaying result
4. **print**(info)

Output:

```
Welcome to Python 3.5's help utility!
```

Python min() Function

Python **min()** function is used to get the smallest element from the collection. This function takes two arguments, first is a collection of elements and second is key, and returns the smallest element from the collection.

Python min() Function Example

1. # Calling function
2. small = min(2225,325,2025) # returns smallest element
3. small2 = min(1000.25,2025.35,5625.36,10052.50)
4. # Displaying result
5. **print**(small)
6. **print**(small2)

Output:

```
325
1000.25
```

Python set() Function

In python, a set is a built-in class, and this function is a constructor of this class. It is used to create a new set using elements passed during the call. It takes an iterable object as an argument and returns a new set object.

Python set() Function Example

```
# Calling function
result = set() # empty set
result2 = set('12')
result3 = set('javatpoint')
# Displaying result
print(result)
print(result2)
print(result3)
```

Output:

```
set()
{'1', '2'}
{'a', 'n', 'v', 't', 'j', 'p', 'i', 'o'}
```

Python hex() Function

Python **hex()** function is used to generate hex value of an integer argument. It takes an integer argument and returns an integer converted into a hexadecimal string. In case, we want to get a hexadecimal value of a float, then use float.hex() function.

Python hex() Function Example

```
# Calling function
result = hex(1)
# integer value
result2 = hex(342)
# Displaying result
print(result)
print(result2)
```

Output:

```
0x1
0x156
```

Python id() Function

Python **id()** function returns the identity of an object. This is an integer which is guaranteed to be unique. This function takes an argument as an object and returns a unique integer number which represents identity. Two objects with non-overlapping lifetimes may have the same id() value.

Python id() Function Example

```
# Calling function
val = id("Javatpoint") # string object
val2 = id(1200) # integer object
val3 = id([25,336,95,236,92,3225]) # List object
# Displaying result
print(val)
print(val2)
print(val3)
```

Output:

```
139963782059696
139963805666864
139963781994504
```

Python setattr() Function

Python **setattr()** function is used to set a value to the object's attribute. It takes three arguments, i.e., an object, a string, and an arbitrary value, and returns none. It is helpful when we want to add a new attribute to an object and set a value to it.

Python setattr() Function Example

```
class Student:
    id = 0
    name = ""

    def __init__(self, id, name):
        self.id = id
        self.name = name

student = Student(102,"Sohan")
print(student.id)
print(student.name)
#print(student.email) product error
setattr(student, 'email','sohan@abc.com') # adding new attribute
print(student.email)
```

Output:

```
102
Sohan
sohan@abc.com
```

Python slice() Function

Python **slice()** function is used to get a slice of elements from the collection of elements. Python provides two overloaded slice functions. The first function takes a single argument while the second function takes three arguments and returns a slice object. This slice object can be used to get a subsection of the collection.

Python slice() Function Example

```
# Calling function
result = slice(5) # returns slice object
result2 = slice(0,5,3) # returns slice object
# Displaying result
print(result)
print(result2)
```

Output:

```
slice(None, 5, None)
slice(0, 5, 3)
```

Python sorted() Function

Python **sorted()** function is used to sort elements. By default, it sorts elements in an ascending order but can be sorted in descending also. It takes four arguments and returns a collection in sorted order. In the case of a dictionary, it sorts only keys, not values.

Python sorted() Function Example

```
str = "javatpoint" # declaring string
# Calling function
sorted1 = sorted(str) # sorting string
# Displaying result
print(sorted1)
```

Output:

```
['a', 'a', 'i', 'j', 'n', 'o', 'p', 't', 't', 'v']
```

Python next() Function

Python **next()** function is used to fetch next item from the collection. It takes two arguments, i.e., an iterator and a default value, and returns an element.

This method calls on iterator and throws an error if no item is present. To avoid the error, we can set a default value.

Python next() Function Example

```
number = iter([256, 32, 82]) # Creating iterator
# Calling function
item = next(number)
# Displaying result
print(item)
# second item
item = next(number)
print(item)
# third item
item = next(number)
print(item)
```

Output:

```
256
32
82
```

Python input() Function

Python **input()** function is used to get an input from the user. It prompts for the user input and reads a line. After reading data, it converts it into a string and returns it. It throws an error **EOFError** if EOF is read.

Python input() Function Example

```
# Calling function
val = input("Enter a value: ")
# Displaying result
print("You entered:",val)
```

Output:

```
Enter a value: 45
You entered: 45
```

Python int() Function

Python **int()** function is used to get an integer value. It returns an expression converted into an integer number. If the argument is a floating-point, the conversion truncates the number. If the argument is outside the integer range, then it converts the number into a long type.

If the number is not a number or if a base is given, the number must be a string.

Python int() Function Example

```
# Calling function
val = int(10) # integer value
val2 = int(10.52) # float value
val3 = int('10') # string value
# Displaying result
print("integer values :",val, val2, val3)
```

Output:

```
integer values : 10 10 10
```

Python isinstance() Function

Python **isinstance()** function is used to check whether the given object is an instance of that class. If the object belongs to the class, it returns true. Otherwise returns False. It also returns true if the class is a subclass.

The **isinstance()** function takes two arguments, i.e., object and classinfo, and then it returns either True or False.

Python isinstance() function Example

```
class Student:
    id = 101
    name = "John"
    def __init__(self, id, name):
        self.id=id
        self.name=name

student = Student(1010,"John")
lst = [12,34,5,6,767]
# Calling function
```

```
print(isinstance(student, Student)) # isinstance of Student class
print(isinstance(lst, Student))
```

Output:

```
True
False
```

Python oct() Function

Python **oct()** function is used to get an octal value of an integer number. This method takes an argument and returns an integer converted into an octal string. It throws an error **TypeError**, if argument type is other than an integer.

Python oct() function Example

```
# Calling function
val = oct(10)
# Displaying result
print("Octal value of 10:",val)
```

Output:

```
Octal value of 10: 0o12
```

Python ord() Function

The python **ord()** function returns an integer representing Unicode code point for the given Unicode character.

Python ord() function Example

```
# Code point of an integer
print(ord('8'))

# Code point of an alphabet
print(ord('R'))

# Code point of a character
print(ord('&'))
```

Output:

```
56
```

```
82
38
```

Python pow() Function

The python **pow()** function is used to compute the power of a number. It returns x to the power of y. If the third argument(z) is given, it returns x to the power of y modulus z, i.e. (x, y) % z.

Python pow() function Example

```
# positive x, positive y (x**y)
print(pow(4, 2))

# negative x, positive y
print(pow(-4, 2))

# positive x, negative y (x**-y)
print(pow(4, -2))

# negative x, negative y
print(pow(-4, -2))
```

Output:

```
16
16
0.0625
0.0625
```

Python print() Function

The python **print()** function prints the given object to the screen or other standard output devices.

Python print() function Example

```
print("Python is programming language.")

x = 7
# Two objects passed
print("x =", x)

y = x
```

```
# Three objects passed
print('x =', x, '= y')
```

Output:

```
Python is programming language.
x = 7
x = 7 = y
```

Python range() Function

The python **range()** function returns an immutable sequence of numbers starting from 0 by default, increments by 1 (by default) and ends at a specified number.

Python range() function Example

```
# empty range
print(list(range(0)))

# using the range(stop)
print(list(range(4)))

# using the range(start, stop)
print(list(range(1,7 )))
```

Output:

```
[]
[0, 1, 2, 3]
[1, 2, 3, 4, 5, 6]
```

Python reversed() Function

The python **reversed()** function returns the reversed iterator of the given sequence.

Python reversed() function Example

```
# for string
String = 'Java'
print(list(reversed(String)))

# for tuple
Tuple = ('J', 'a', 'v', 'a')
print(list(reversed(Tuple)))
```

```

# for range
Range = range(8, 12)
print(list(reversed(Range)))

# for list
List = [1, 2, 7, 5]
print(list(reversed(List)))
```

Output:

```
['a', 'v', 'a', 'J']
['a', 'v', 'a', 'J']
[11, 10, 9, 8]
[5, 7, 2, 1]
```

Python round() Function

The python **round()** function rounds off the digits of a number and returns the floating point number.

Python round() Function Example

```
#  for integers
print(round(10))

#  for floating point
print(round(10.8))

#  even choice
print(round(6.6))
```

Output:

```
10
11
7
```

Python issubclass() Function

The python **issubclass()** function returns true if object argument(first argument) is a subclass of second class(second argument).

Python issubclass() Function Example

```
class Rectangle:
  def __init__(rectangleType):
    print('Rectangle is a ', rectangleType)

class Square(Rectangle):
  def __init__(self):
    Rectangle.__init__('square')

print(issubclass(Square, Rectangle))
print(issubclass(Square, list))
print(issubclass(Square, (list, Rectangle)))
print(issubclass(Rectangle, (list, Rectangle)))
```

Output:

```
True
False
True
True
```

Python str

The python **str()** converts a specified value into a string.

Python str() Function Example

```
str('4')
```

Output:

```
'4'
```

Python tuple() Function

The python **tuple()** function is used to create a tuple object.

Python tuple() Function Example

```
t1 = tuple()
print('t1=', t1)

# creating a tuple from a list
t2 = tuple([1, 6, 9])
print('t2=', t2)

```

```
# creating a tuple from a string
t1 = tuple('Java')
print('t1=',t1)

# creating a tuple from a dictionary
t1 = tuple({4: 'four', 5: 'five'})
print('t1=',t1)
```

Output:

```
t1= ()
t2= (1, 6, 9)
t1= ('J', 'a', 'v', 'a')
t1= (4, 5)
```

Python type()

The python **type()** returns the type of the specified object if a single argument is passed to the type() built in function. If three arguments are passed, then it returns a new type object.

Python type() Function Example

```
List = [4, 5]
print(type(List))

Dict = {4: 'four', 5: 'five'}
print(type(Dict))

class Python:
    a = 0

InstanceOfPython = Python()
print(type(InstanceOfPython))
```

Output:

```
<class 'list'>
<class 'dict'>
<class '__main__.Python'>
```

Python vars() function

The python **vars()** function returns the __dict__ attribute of the given object.

Python vars() Function Example

```
class Python:
  def __init__(self, x = 7, y = 9):
    self.x = x
    self.y = y

InstanceOfPython = Python()
print(vars(InstanceOfPython))
```

Output:

```
{'y': 9, 'x': 7}
```

Python zip() Function

The python **zip()** Function returns a zip object, which maps a similar index of multiple containers. It takes iterables (can be zero or more), makes it an iterator that aggregates the elements based on iterables passed, and returns an iterator of tuples.

Python zip() Function Example

```
numList = [4,5, 6]
strList = ['four', 'five', 'six']

# No iterables are passed
result = zip()

# Converting itertor to list
resultList = list(result)
print(resultList)

# Two iterables are passed
result = zip(numList, strList)

# Converting itertor to set
resultSet = set(result)
print(resultSet)
```

Output:

```
[]
{(5, 'five'), (4, 'four'), (6, 'six')}
```

Views and their use

In relational databases, data is structured using various database objects like tables, stored procedure, views, clusters etc. This article aims to walk you through 'SQL VIEW' – one of the widely-used database objects in SQL Server.

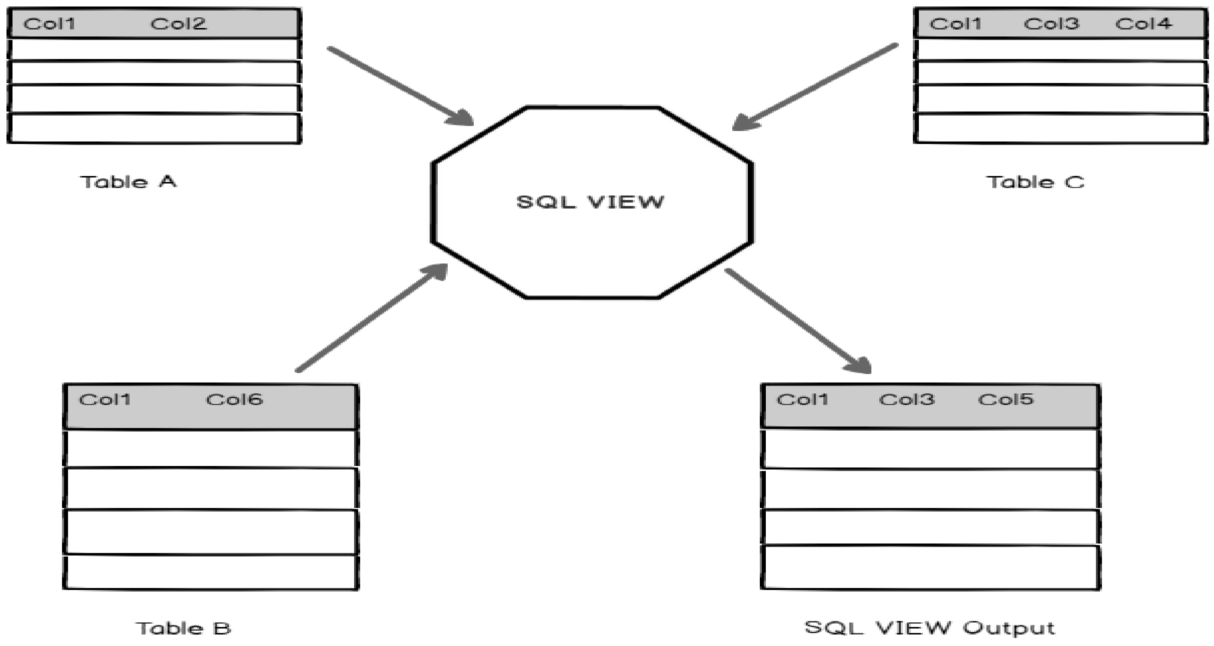
It is a good practice to organize tables in a database to reduce redundancy and dependency in SQL database. Normalization is a database process for organizing the data in the database by splitting large tables into smaller tables. These multiple tables are linked using the relationships. Developers write queries to retrieve data from multiple tables and columns. In the query, we might use multiple joins and queries could become complicated and overwhelming to understand. Users should also require permissions on individual objects to fetch the data.

Let's go ahead and see how SQL VIEW help to resolve these issues in SQL Server.

Introduction

A VIEW in SQL Server is like a virtual table that contains data from one or multiple tables. It does not hold any data and does not exist physically in the database. Similar to a SQL table, the view name should be unique in a database. It contains a set of predefined SQL queries to fetch data from the database. It can contain database tables from single or multiple databases as well.

In the following image, you can see the VIEW contains a query to join three relational tables and fetch the data in a virtual table.



A VIEW does not require any storage in a database because it does not exist physically. In a VIEW, we can also control user security for accessing the data from the database tables. We can allow users to get the data from the VIEW, and the user does not require permission for each table or column to fetch data.

Let’s explore user-defined VIEW in SQL Server.

Note: In this article, I am going to use sample database **AdventureWorks** for all examples.

Create a SQL VIEW

The syntax to create a VIEW is as follows:

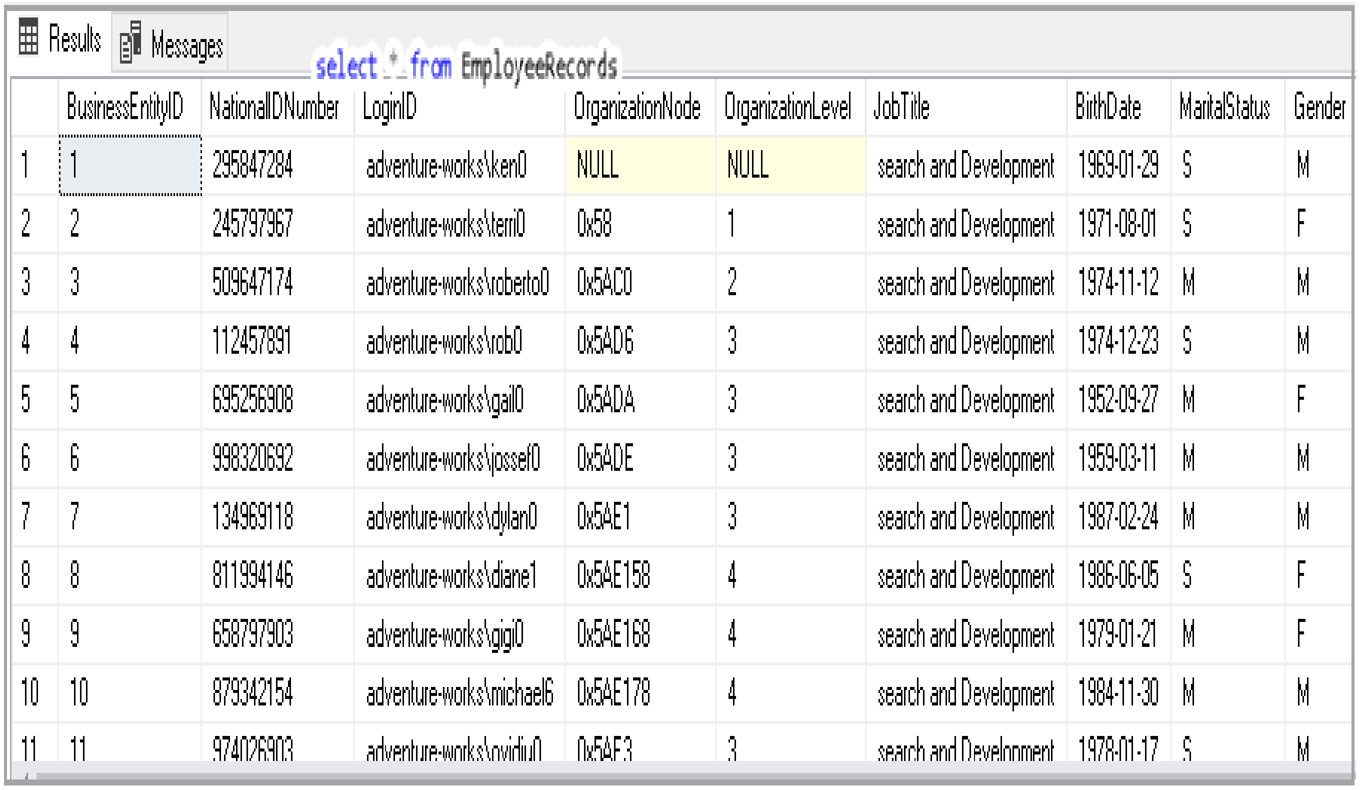
```
CREATE VIEW Name AS

Select column1, Column2...Column N From tables

Where conditions;
```

Example 1: SQL VIEW to fetch all records of a table

It is the simplest form of a VIEW. Usually, we do not use a VIEW in SQL Server to fetch all records from a single table.

```
CREATE VIEW EmployeeRecords

AS

    SELECT *

    FROM [HumanResources].[Employee];
```

Once a VIEW is created, you can access it like a SQL table.



```
select * from EmployeeRecords
```

	BusinessEntityID	NationalIDNumber	LoginID	OrganizationNode	OrganizationLevel	JobTitle	BirthDate	MaritalStatus	Gender
1	1	295847284	adventure-works\ken0	NULL	NULL	search and Development	1969-01-29	S	M
2	2	245797967	adventure-works\terri0	0x58	1	search and Development	1971-08-01	S	F
3	3	509647174	adventure-works\roberto0	0x5AC0	2	search and Development	1974-11-12	M	M
4	4	112457891	adventure-works\rob0	0x5AD6	3	search and Development	1974-12-23	S	M
5	5	695256908	adventure-works\gail0	0x5ADA	3	search and Development	1952-09-27	M	F
6	6	998320692	adventure-works\jossef0	0x5ADE	3	search and Development	1959-03-11	M	M
7	7	134969118	adventure-works\dylan0	0x5AE1	3	search and Development	1987-02-24	M	M
8	8	811994146	adventure-works\diane1	0x5AE158	4	search and Development	1986-06-05	S	F
9	9	658797903	adventure-works\gigi0	0x5AE168	4	search and Development	1979-01-21	M	F
10	10	879342154	adventure-works\michael6	0x5AE178	4	search and Development	1984-11-30	M	M
11	11	974026903	adventure-works\ovidiu0	0x5AE3	3	search and Development	1978-01-17	S	M

Example 2: SQL VIEW to fetch a few columns of a table

We might not be interested in all columns of a table. We can specify required column names in the select statement to fetch those fields only from the table.

```
CREATE VIEW EmployeeRecords

AS

SELECT NationalIDNumber,LoginID,JobTitle

FROM [HumanResources].[Employee];
```

Example 3: SQL VIEW to fetch a few columns of a table and filter results using WHERE clause

We can filter the results using a Where clause condition in a Select statement. Suppose we want to get EmployeeRecords with Martial status ‘M’.

```
CREATE VIEW EmployeeRecords
AS
  SELECT NationalIDNumber,
         LoginID,
         JobTitle,
         MaritalStatus
  FROM [HumanResources].[Employee]
  WHERE MaritalStatus = 'M';
```

Example 4: SQL VIEW to fetch records from multiple tables

We can use VIEW to have a select statement with Join condition between multiple tables. It is one of the frequent uses of a VIEW in SQL Server.

In the following query, we use INNER JOIN and LEFT OUTER JOIN between multiple tables to fetch a few columns as per our requirement.

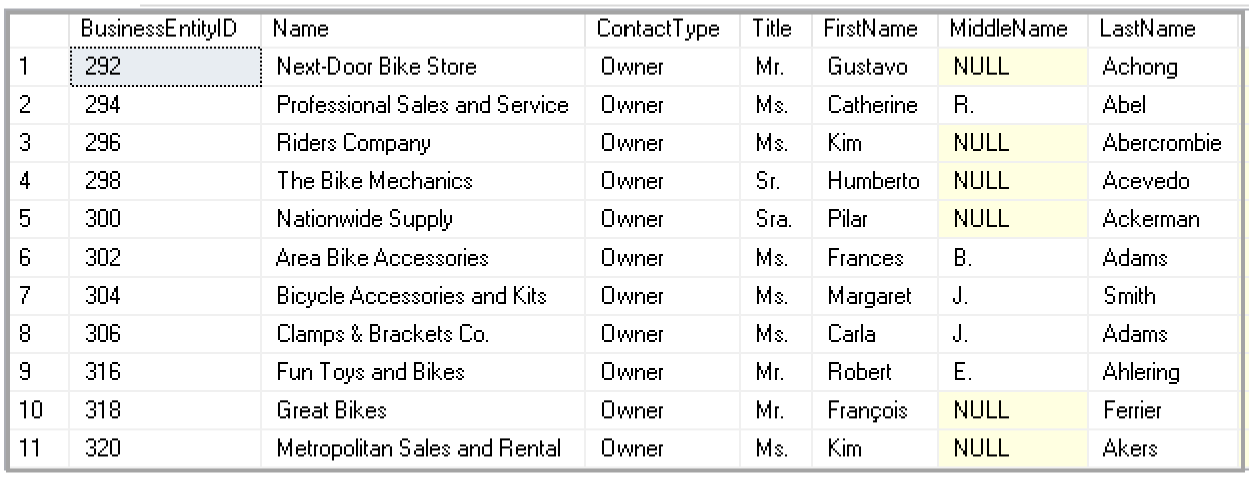
```
CREATE VIEW [Sales].[vStoreWithContacts]
AS
    SELECT s.[BusinessEntityID],
           s.[Name],
           ct.[Name] AS [ContactType],
```

```sql
        p.[Title],
        p.[FirstName],
        p.[MiddleName],
        p.[LastName],
        p.[Suffix],
        pp.[PhoneNumber],
        ea.[EmailAddress],
        p.[EmailPromotion]
    FROM [Sales].[Store] s
        INNER JOIN [Person].[BusinessEntityContact] bec ON bec.[BusinessEntityID] = s
.      [BusinessEntityID]
        INNER JOIN [Person].[ContactType] ct ON ct.[ContactTypeID] = bec.[ContactTyp
eID]
        INNER JOIN [Person].[Person] p ON p.[BusinessEntityID] = bec.[PersonID]
        LEFT OUTER JOIN [Person].[EmailAddress] ea ON ea.[BusinessEntityID] = p.[Bu
sinessEntityID]
        LEFT OUTER JOIN [Person].[PersonPhone] pp ON pp.[BusinessEntityID] = p.[Bu
sinessEntityID];
```

GO

Suppose you need to execute this query very frequently. Using a VIEW, we can simply get the data with a single line of code.

```
select * from  [Sales].[vStoreWithContacts]
```



	BusinessEntityID	Name	ContactType	Title	FirstName	MiddleName	LastName
1	292	Next-Door Bike Store	Owner	Mr.	Gustavo	NULL	Achong
2	294	Professional Sales and Service	Owner	Ms.	Catherine	R.	Abel
3	296	Riders Company	Owner	Ms.	Kim	NULL	Abercrombie
4	298	The Bike Mechanics	Owner	Sr.	Humberto	NULL	Acevedo
5	300	Nationwide Supply	Owner	Sra.	Pilar	NULL	Ackerman
6	302	Area Bike Accessories	Owner	Ms.	Frances	B.	Adams
7	304	Bicycle Accessories and Kits	Owner	Ms.	Margaret	J.	Smith
8	306	Clamps & Brackets Co.	Owner	Ms.	Carla	J.	Adams
9	316	Fun Toys and Bikes	Owner	Mr.	Robert	E.	Ahlering
10	318	Great Bikes	Owner	Mr.	François	NULL	Ferrier
11	320	Metropolitan Sales and Rental	Owner	Ms.	Kim	NULL	Akers

Example 5: SQL VIEW to fetch specific column

In the previous example, we created a VIEW with multiple tables and a few column from those tables. Once we have a view, it is not required to fetch all columns from the view. We can select few columns as well from a VIEW in SQL Server similar to a relational table.

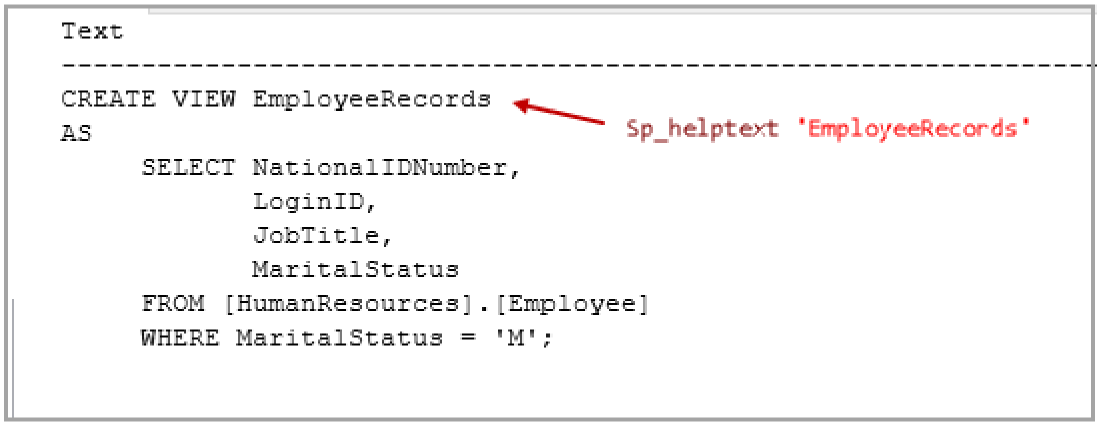
In the following query, we want to get only two columns name and contract type from the view.

```
SELECT Name,

       ContactType

FROM [Sales].[vStoreWithContacts];
```

Example 6: Use Sp_helptext to retrieve VIEW definition

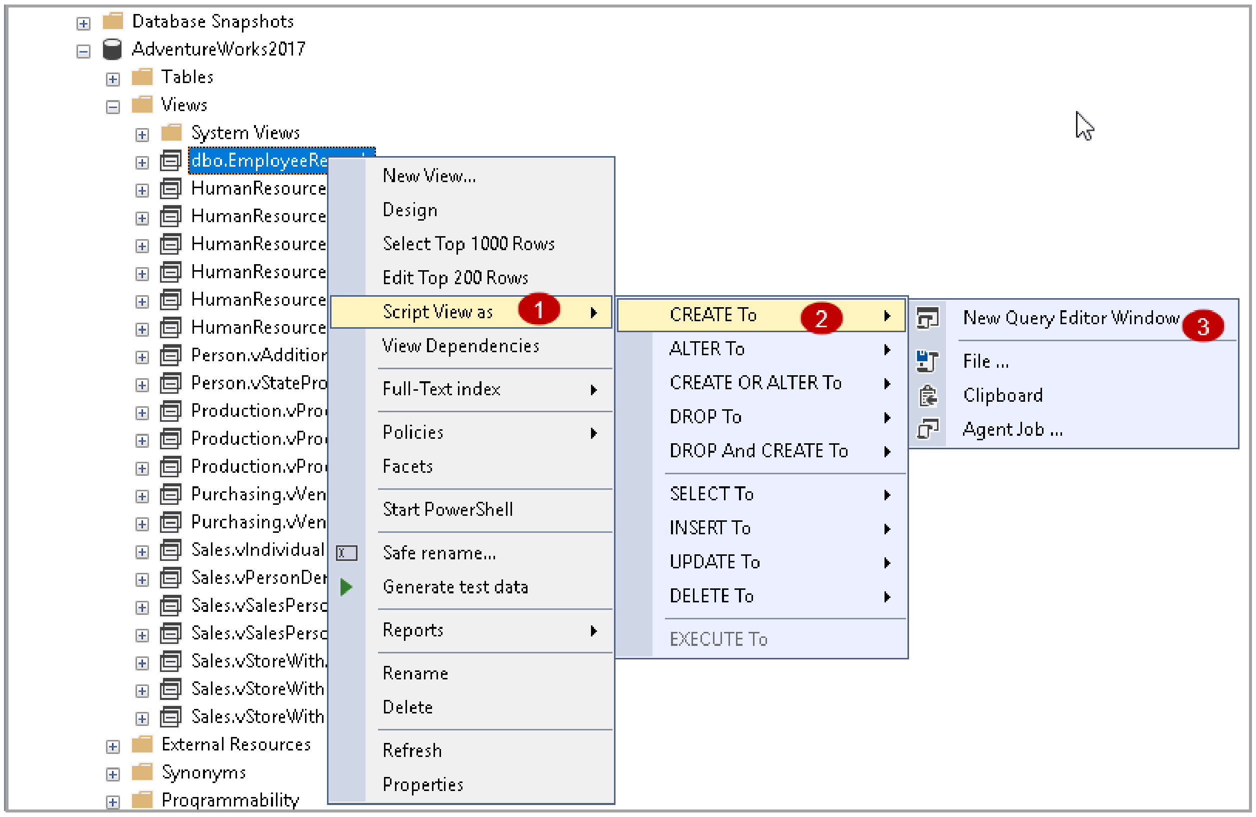
We can use sp_helptext system stored procedure to get VIEW definition. It returns the complete definition of a SQL VIEW.

For example, let’s check the view definition for EmployeeRecords VIEW.



```
Text
------------------------------------------------------------------
CREATE VIEW EmployeeRecords
AS
     SELECT NationalIDNumber,
            LoginID,
            JobTitle,
            MaritalStatus
     FROM [HumanResources].[Employee]
     WHERE MaritalStatus = 'M';
```

We can use SSMS as well to generate the script for a VIEW. Expand database -> Views -> Right click and go to Script view as -> Create To -> New Query Editor Window.



Overviews of ORACLE

If you are a user of Oracle E-Business Suite and are continuing to have problems with your master data, then it probably goes back to the time when you first implemented it. It is not an uncommon problem. Most systems integrators did not pay too much attention to data quality. However, you are now left holding the bag. You are the one that is dealing with the consequences of poor data quality in your day to day operations. It is not too late. Read about our story to understand why you can still fix the problem. It would help you appreciate why an MDM program will NOT cost you an arm and a leg and it will help you build a business case much easier than to justify a seven-figure investment. It will also help you appreciate that if your waiting for the Cloud to magically solve all problems, then it is not going to happen. If you solve the problem now, it will also help you move to the cloud.

Our story started in 1995 when we were tasked with implementing Oracle ERP at Lucent Microelectronics (AT&T Microelectronics then) in Allentown, PA. We were required to integrate all their internal factories, foundries and Sub Contractors into a single instance to be the single source of truth for the entire company.

While Oracle has the deep capability for business processes and inter department integration, it is easily susceptible to bad data. That would destroy the objective of the entire ERP implementation. Good data is at the heart of timely, reliable, accurate, and complete information and is the bedrock of data-oriented decision making. Good Master data, is a prerequisite for good data. To generate good master data, the following attributes are mandatory.

1. Easy User Interface
2. Real time error checking to enforce policies and constraints
3. Approvals to ensure stakeholders signoff
4. No Latency between the time the data is created and its consumption in downstream transactions
5. No gaps between data in the ERP and data in the MDM.

To meet these objectives, we built a custom solution for them and it was for the product domain. Subsequently, when we implemented the same ERP for Sony Semiconductor in Japan, we decided to build a framework that could handle multiple business entities such as Product, Customer, Supplier, GL account, Cost Center, Location, People etc.,

Since Sony Semiconductor went live, many other customers have used Triniti's MDM to create and maintain high quality master data. They include Qualcomm, Power Integrations, DSPG and Peregrine Semiconductor.

With Triniti's MDM you can achieve the benefits of zero latency enterprise to make faster, and reliable decision making. Armed with error-free transactions in your ERP,CRM, etc, you will be able to avoid pitfalls of not meeting program expectations.

By using Triniti's MDM you can achieve the following functionality:

- Model your master data for applications other than Oracle and SFDC as well
- Model additional complex hierarchies that represent your master data
- Set your own policies and constraints, other than that are required by Oracle and SFDC which are already builtin
- Validate foreign keys with member applications directly without replicating in the MDM
- Do data quality checks
- Integrate with other applications in real time using industry standard protocols
- Enforce authorship
- Execute survivorship in member applications
- Define required workflows and approvals
- Get data quality metrics
- Configure 360-degree views of your domains

Summary

Since we have solved the problem, you can leverage our experience and our tools. The integration is 100% reusable and is robust. The backbone is a framework and not a custom solution. It has continued to evolve as technology has evolved, that is illustrated by our ability to integrate with SFDC and demonstrate that we can quickly adapt to the Cloud. This is what enables you to acquire industrial strength MDM capabilities at a very affordable price from us. We also save you considerable time during implementation by reading YOUR configuration and not just configuration based on the VISION database. If you do not solve the problem now, then not only will you continue to bear the consequences of poor data quality, but you will either carry it to the cloud, or will impede your move to the cloud. On the other hand if you do it now, then your move to the cloud will be clean. You would stilll have to worry about MDM being a part of cloud, but if you go with us, we will ensure that we are integrated to your cloud platform as well.

Triniti MDM provides out-of-the-box support for Oracle E-Business suite and SFDC for Product, Customer, Supplier, and GL Chart of accounts structure including GL Account and Cost Center. Check back! We will update our content as we release more domains.

To make your job easier, we also provide the following tools:

1. An ROI Calculator.
2. A Vendor evaluation form.

Data definition and manipulation

DBMS software primarily functions as an interface between the end user and the database, simultaneously managing the data, the database engine, and the database schema in order to facilitate the organization and manipulation of data.

Though functions of DBMS vary greatly, general-purpose DBMS features and capabilities should include: a user accessible catalog describing metadata, DBMS library management system, data abstraction and independence, data security, logging and auditing of activity, support for concurrency and transactions, support for authorization of access, access support from remote locations, DBMS data recovery support in the event of damage, and enforcement of constraints to ensure the data follows certain rules.

A database schema design technique that functions to increase clarity in organizing data is referred to as normalization. Normalization in DBMS modifies an existing schema to minimize redundancy and dependency of data by splitting a large table into smaller tables and defining the relationship between them. DBMS Output is a built-in package SQL in DBMS that enables the user to display debugging information and output, and send messages from subprograms, packages, PL/SQL blocks, and triggers. Oracle originally developed the DBMS File Transfer package, which provides procedures to copy a binary file within a database or to transfer a binary file between databases.

A database management system functions through the use of system commands, first receiving instructions from a database administrator in DBMS, then instructing the system accordingly, either to retrieve data, modify data, or load existing data from the system. Popular DBMS examples include cloud-based database management systems, in-memory database management systems (IMDBMS), columnar database management systems (CDBMS), and NoSQL in DBMS.

RDBMS vs DBMS

A relational database management system (RDBMS) refers to a collection of programs and capabilities that is designed to enable the user to create, update, and administer a relational database, which is characterized by its structuring of data into logically independent tables. There are several features that distinguish a Relational DBMS from a DBMS, including:

- Structure: Where data is structured in hierarchical form in a DBMS, data is structured in tabular form in a RDBMS.

- User capacity: A RDBMS is capable of operating with multiple users. DBMS can only manage one user at a time.

- Software/hardware requirements: A RDBMS has greater software and hardware requirements.

- Programs managed: DBMS maintains databases within the computer network and system hard disks. A RDBMS manages the relationships between its incorporated tables of data.

- Data capacity: A DBMS is capable of managing small amounts of data and a RDBMS can manage an unlimited amount of data.

- Distributed databases: A DBMS does not provide support for distributed databases while a RDBMS does.

- ACID implementation: A RDBMS bases the structure of its data on the ACID (Atomicity, Consistency, Isolation, and Durability) model.

Difference Between Data and Information in DBMS

Data is raw, unprocessed, unorganized facts that are seemingly random and do not yet carry any significance or meaning. Information refers to data that has been organized, interpreted, and contextualized by a human or machine so that it possess relevance and purpose.

Information is filtered data that has been made systematic and useful, and is considered to be more reliable and valuable to researchers as proper analysis and refinement has been conducted. A DBMS is concerned with the manipulation of data in a database.

Difference Between Data Models in DBMS

A data model is an abstract model that organizes elements of data, documents the way data is stored and retrieved, standardizes how different data elements relate to one another and to the properties of real-world entities, and designs the responses needed for information system requirements. There are three main types of DBMS data models: relational, network, and hierarchical.

- Relational data model: Data is organized as logically independent tables.
- Network data model: All entities are organized in graphical representations.
- Hierarchical data model: Data is organized into a tree-like structure.

Other data models include entity-relationship, record base, object-oriented, object relation, semi-structured, associative, context, and flat data models. Database system architecture in DBMS is categorized as either single tier, in which the DBMS is the only entity where the user directly sits on the DBMS and uses it, or multi-tier, in which nearly all components are independent and can be changed independently.

Features of Distributed Database Management System

A distributed database is a collection of related data in multiple interconnected databases that are logically interrelated, but physically stored across multiple physical locations. Distributed databases are categorized as either homogeneous, in which all the physical locations use the same hardware and run the same operating systems and applications, or heterogeneous, in which each location may have different data, software, and hardware structures.

A distributed database management system (DDBMS) refers to a centralized application that functions to create and manipulate distributed databases, synchronize the database at regular intervals and provide transparent access mechanisms to the user, ensure universal application of data modifications, maintain data security and integrity of the database, can be accessed by several users simultaneously, and is used in applications that process large volumes of data.

How is a DBMS Different from a Traditional File System?

A traditional filing system refers to early endeavors to computerize the manual filing system. File-based systems typically use storage devices such as a CD-ROM or hard disk to store and organize computer files and the data within with the goal of facilitating easy access.

A traditional file system is inexpensive, ideal for a small system with smaller quantity of parts, very low design efforts, isolated data, and has a simple backup system, but is not secure, has a lack of flexibility and many limitations, and has integrity flaws.

The benefits of DBMS over a traditional file system include: good for large systems, data-sharable, flexible, has data integrity, and has a complex backup system. DBMS data security requirements leverage the use of masking, tokenization, encryption, access control lists, permissions, firewalls, and virtual private networks, making data storage and querying in DBMS a far more secure option than in a traditional file system.

Does OmniSci Offer a DBMS Solution?

The analytics platform is the solution designed to compensate for the inadequacies of the relational database management system, working in tandem with various data processing techniques to address the increasing demands of users in large, data-driven industries. While so much of today’s data is now location-enriched, geospatial-specific processes in GIS tools are becoming too slow for today's data volumes. OmniSci bridges this divide by making geospatial intelligence (GEOINT) capabilities a first-class citizen of our accelerated analytics platform.

Unit - V

Database Security, Integrity and Control

Security and Integrity threats

In this chapter, we will look into the threats that a database system faces and the measures of control. We will also study cryptography as a security tool.

Database Security and Threats

Data security is an imperative aspect of any database system. It is of particular importance in distributed systems because of large number of users, fragmented and replicated data, multiple sites and distributed control.

Threats in a Database

- **Availability loss −** Availability loss refers to non-availability of database objects by legitimate users.
- **Integrity loss −** Integrity loss occurs when unacceptable operations are performed upon the database either accidentally or maliciously. This may happen while creating, inserting, updating or deleting data. It results in corrupted data leading to incorrect decisions.
- **Confidentiality loss −** Confidentiality loss occurs due to unauthorized or unintentional disclosure of confidential information. It may result in illegal actions, security threats and loss in public confidence.

Measures of Control

The measures of control can be broadly divided into the following categories −

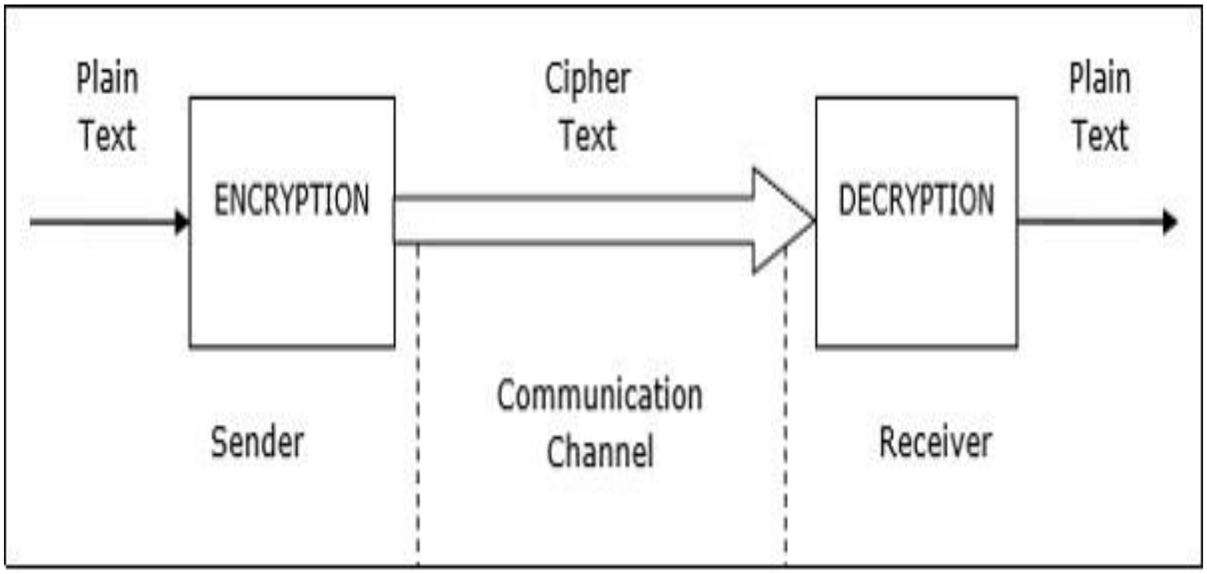
- **Access Control −** Access control includes security mechanisms in a database management system to protect against unauthorized access. A user can gain access to the database after clearing the login process through only valid user accounts. Each user account is password protected.
- **Flow Control −** Distributed systems encompass a lot of data flow from one site to another and also within a site. Flow control prevents data from being transferred in such a way that it can be accessed by unauthorized agents. A flow policy lists out the channels through which information can flow. It also defines security classes for data as well as transactions.
- **Data Encryption −** Data encryption refers to coding data when sensitive data is to be communicated over public channels. Even if an unauthorized agent gains access of the data, he cannot understand it since it is in an incomprehensible format.

What is Cryptography?

Cryptography is the science of encoding information before sending via unreliable communication paths so that only an authorized receiver can decode and use it.

The coded message is called **cipher text** and the original message is called **plain text**. The process of converting plain text to cipher text by the sender is called encoding or **encryption**. The process of converting cipher text to plain text by the receiver is called decoding or **decryption**.

The entire procedure of communicating using cryptography can be illustrated through the following diagram −



Conventional Encryption Methods

In conventional cryptography, the encryption and decryption is done using the same secret key. Here, the sender encrypts the message with an encryption algorithm using a copy of the secret key. The encrypted message is then send over public communication channels. On receiving the encrypted message, the receiver decrypts it with a corresponding decryption algorithm using the same secret key.

Security in conventional cryptography depends on two factors −

- A sound algorithm which is known to all.
- A randomly generated, preferably long secret key known only by the sender and the receiver.

The most famous conventional cryptography algorithm is **Data Encryption Standard** or **DES**.

The advantage of this method is its easy applicability. However, the greatest problem of conventional cryptography is sharing the secret key between the communicating parties. The ways to send the key are cumbersome and highly susceptible to eavesdropping.

Public Key Cryptography

In contrast to conventional cryptography, public key cryptography uses two different keys, referred to as public key and the private key. Each user generates the pair of public key and private key. The user then puts the public key in an accessible place. When a sender wants to sends a message, he encrypts it using the public key of the receiver. On receiving the encrypted message, the receiver decrypts it using his private key. Since the private key is not known to anyone but the receiver, no other person who receives the message can decrypt it.

The most popular public key cryptography algorithms are **RSA** algorithm and **Diffie–Hellman** algorithm. This method is very secure to send private messages. However, the problem is, it involves a lot of computations and so proves to be inefficient for long messages.

The solution is to use a combination of conventional and public key cryptography. The secret key is encrypted using public key cryptography before sharing between the communicating parties. Then, the message is send using conventional cryptography with the aid of the shared secret key.

Digital Signatures

A Digital Signature (DS) is an authentication technique based on public key cryptography used in e-commerce applications. It associates a unique mark to an individual within the body of his message. This helps others to authenticate valid senders of messages.

Typically, a user’s digital signature varies from message to message in order to provide security against counterfeiting. The method is as follows −

- The sender takes a message, calculates the message digest of the message and signs it digest with a private key.
- The sender then appends the signed digest along with the plaintext message.
- The message is sent over communication channel.
- The receiver removes the appended signed digest and verifies the digest using the corresponding public key.
- The receiver then takes the plaintext message and runs it through the same message digest algorithm.
- If the results of step 4 and step 5 match, then the receiver knows that the message has integrity and authentic.

Defense mechanism

Database security encompasses a range of security controls designed to protect the Database Management System (DBMS). The types of database security measures your business should use include protecting the underlying infrastructure that houses the database such as the network and servers), securely configuring the DBMS, and the access to the data itself.

Database security controls

Database security encompasses multiple controls, including system hardening, access, DBMS configuration, and security monitoring. These different security controls help to manage the circumventing of security protocols.

System hardening and monitoring

The underlying architecture provides additional access to the DBMS. It is vital that all systems are patched consistently, hardened using known security configuration standards, and monitored for access, including insider threats.

DBMS configuration

It is critical that the DBMS be properly configured and hardened to take advantage of security features and limit privileged access that may cause a misconfiguration of expected security settings. Monitoring the DBMS configuration and ensuring proper change control processes helps ensure that the configuration stays consistent.

Authentication

Database security measures include authentication, the process of verifying if a user's credentials match those stored in your database, and permitting only authenticated users access to your data, networks, and database platform.

Access

A primary outcome of database security is the effective limitation of access to your data. Access controls authenticate legitimate users and applications, limiting what they can access in your database. Access includes designing and granting appropriate user attributes and roles and limiting administrative privileges.

Database auditing

Monitoring (or auditing) actions as part of a database security protocol delivers centralized oversight of your database. Auditing helps to detect, deter, and reduce the overall impact of unauthorized access to your DBMS.

Backups

A data backup, as part of your database security protocol, makes a copy of your data and stores it on a separate system. This backup allows you to recover lost data that may result from hardware failures, data corruption, theft, hacking, or natural disasters.

Encryption

Database security can include the secure management of encryption keys, protection of the encryption system, management of a secure, off-site encryption backup, and access restriction protocols.

Application security

Database and application security framework measures can help protect against common known attacker exploits that can circumvent access controls, including SQL injection.

Why is database security important?

Safeguarding the data your company collects and manages is of utmost importance. Database security can guard against a compromise of your database, which can lead to financial loss, reputation damage, consumer confidence disintegration, brand erosion, and non-compliance of government and industry regulation.

Database security safeguards defend against a myriad of security threats and can help protect your enterprise from:

- Deployment failure
- Excessive privileges
- Privilege abuse
- Platform vulnerabilities
- Unmanaged sensitive data
- Backup data exposure

- Weak authentication
- Database injection attacks

Auditing and Control

A simple definition for what a database management system (DBMS) is, would be that it is a complex set of software programs that control the organization, storage and retrieval of data in a database. It also controls the security and integrity of the database.

This article will not attempt to give a detailed explanation of database technology, rather it will serve to introduce the IT auditor to some of the concepts that will be necessary to be understood and performed to support an audit of a DBMS.

But first, in order to understand DBMS there is some database terminology and definitions you will need to understand:

- Concurrency Control – Refers to the class of controls used in database management systems (DBMS) to ensure that transactions are processed in an atomic, consistent, isolated and durable manner (ACID). This implies that only serial and recoverable schedules are permitted, and that committed transactions are not discarded when undoing aborted transactions.
- Data Structure – The relationships among files in a database and among data items within each file.
- Database – A stored collection of related data needed by organizations and individuals to meet their information processing and retrieval requirements.
- Database Administrator (DBA) – An individual or department responsible for the security and information classification of the shared data stored on a database system. This responsibility includes the design, definition and maintenance of the database.
- Database Specifications – These are the requirements for establishing a database application. They include field definitions, field requirements, and reporting requirements for the individual information in the database.
- Foreign Key – A foreign key is a value that represents a reference to a tuple (a row in a table) containing the matching candidate key value (in the relational theory it would be a candidate key, but in real DBMS implementations it is always the primary key). The problem of ensuring that the database does not include any invalid foreign key values is therefore known as the referential integrity problem. The constraint that values of a given foreign key must match values of the corresponding candidate key is known as a referential constraint. The relation (table) that contains the foreign key is referred as the referencing relation and the relations that contain the corresponding candidate key as the referenced relation or target relation.

- Normalization – The elimination of redundant data.
- Repository – The central database that stores and organizes data.
- Transaction log – A manual or automated log of all updates to data files and databases.
- Tuple – A tuple is a row in a database table.

When we speak about Database Management Systems (DBMS), there are three basic types:

- Hierarchical – a database structured in a tree/root or parent/child relationship. Each parent can have many children; however, each child may have only one parent.
- Network – the basic data modeling construct is called a set. A set is formed by an owner record type, a member record type and a name. A member record type can have that role in more than one set, so a multi-owner relationship is allowed. An owner record type can also be a member or owner in another set. Usually, a set defines a 1:N relationship, although one-to-one (1:1) is permitted. A disadvantage of the network model is that such structures can be extremely complex and difficult to comprehend, modify or reconstruct in case of failure.
- Relational – This model is based on the set theory and relational calculations. A relational database allows the definition of data structures, storage/retrieval operations and integrity constraints. In such a database the data and relationships among these data are organized in tables. A table is a collection of rows, also known as tuples, and each tuple in a table contains the same number of columns. Columns, called domains or attributes, correspond to fields. Tuples are equal to records in a conventional file structure.

Relational tables have the following characteristics:

- Values are atomic
- Each row is unique
- Column values are of the same kind
- The sequence of columns is insignificant
- The sequence of rows is insignificant
- Each column has a unique name

Some of the advantages of the relational model over the hierarchical and network model are that it is easier:

- For users to understand and implement a physical database system
- To convert from other database structures
- To implement projection and join operations
- To create new relations for applications

- To implement access control over sensitive data
- To modify the data base

  When auditing the controls of a database, the auditor would check to see that the following controls have been implemented and maintained to ensure database integrity and availability:

- Definition standards
- Data backup and recovery procedures
- Access controls
- Only authorized personnel can update the database
- Controls to handle concurrent access problems such as multiple users trying to update the same record at the same time
- Controls to ensure the accuracy, completeness and consistency of data elements and relationships.
- Checkpoints to minimize data loss
- Database re-organizations
- Monitoring database performance
- Capacity planning
- Who can access the database without going through the application?

  When we speak of who can access the database, we have already identified one of the major audit concerns and that is what access does the DBA have? As everyone knows the DBA basically has the "keys to the kingdom" and can do (read, write, change, delete) anything. What you have to make sure of is that someone is watching. Someone is monitoring (logging) the actions the DBA takes. And the DBA, doesn't have the ability to de-activate the log nor do they have access to the log.

  It goes without saying that Access Control is the number one issue with database management systems. That being said let's not forget to audit disaster recovery and restoration, patch management, change management, incident logging and all the other issues an auditor should look for.

  There is another issue that auditors need to deal with when auditing DBMS and that is to perform some type of data integrity testing. Data integrity testing is a set of substantive tests (NOTE: Substantive not Compliance testing) that examines accuracy, completeness, consistency and authorization of data presently held in a system. There are two common types of data integrity tests; relational and referential. Relational integrity tests are performed at the data element and record-based levels. It is enforced through data validation routines built into the application or by defining the input condition constraints and data characteristics at the table definition in the database stage. Sometimes it is a combination of both.

Referential integrity test define existence relationships between entities in different tables of a database that needs to be maintained by the DBMS. Referential integrity checks involve ensuring that all references to a primary key from another table actually exist in their original table.

With respect to data integrity in online transaction processing systems there are four online data integrity requirements known collectively as the ACID principle. For those of you that are old enough to remember ACID, congratulations, your brain isn't completed fried.

The "A" stands for atomicity and from a user's perspective, a transaction is either completed in its entirety or not at all.

The "C" stands for consistency. Basically, all integrity conditions in the database are maintained with each transaction, taking the database from one consistent state into another consistent state.

The "I" stands for isolation. Each transaction is isolated from other transactions and hence each transaction only accesses data that are part of a consistent database state.

The "D" stands for durability. If a transaction has been reported back to a user as complete, the resulting changes to the database survive subsequent hardware of software failures.

As a parting comment, I would be remiss, if I didn't mention how the database was populated in a test environment. As many times as I have audited databases, I have found that the production environment was being copied to the test environment to ensure an accurate copy so that changes would not fail once they were moved to production. At least that was the logical in the client's explanation. What they fail to realize is that the security controls in test are significantly weaker that they are in production and yet there is a mirror unprotected copy sitting there for all to see.

At least as an auditor, you should recommend that the data be sanitized before being used in test.

I hope you've enjoyed this brief overview of DBMS and have an appreciation of some things you might check as an auditor.

Recent trends in DBMS

Concepts in database management hardly fall in the category of come-and-go, as the cost of shifting between technical approaches overwhelms producers, managers, and designers. However, there are several trends in database management, and knowing how to take advantage of them will benefit your organization. Following are the some of the current trends:

1. **Databases that bridge SQL/NoSQL**

The latest trends in database products are those that don't simply embrace a single database structure. Instead, the databases bridge SQL and NoSQL, giving users the best capabilities offered by both. This includes products that allow users to access a NoSQL database in the same way as a relational database, for example.

2. **Databases in the cloud/Platform as a Service**
As developers continue pushing their enterprises to the cloud, organizations are carefully weighing the trade-offs associated with public versus private. Developers are also determining how to combine cloud services with existing applications and infrastructure. Providers of cloud service offer many options to database administrators. Making the move towards the cloud doesn't mean changing organizational priorities, but finding products and services that help your group meet its goals.

3. **Automated management**
Automating database management is another emerging trend. The set of such techniques and tools intend to simplify maintenance, patching, provisioning, updates and upgrades — even project workflow. However, the trend may have limited usefulness since database management frequently needs human intervention.

4. **An increased focus on security**
While not exactly a trend given the constant focus on data security, recent ongoing retail database breaches among US-based organizations show with ample clarity the importance for database administrators to work hand-in-hand with their IT security colleagues to ensure all enterprise data remains safe. Any organization that stores data is vulnerable.
Database administrators must also work with the security team to eliminate potential internal weaknesses that could make data vulnerable. These could include issues related to network privileges, even hardware or software misconfigurations that could be misused, resulting in data leaks.

5. In-memory databases
Within the data warehousing community there are similar questions about columnar versus row-based relational tables; the rise of in-memory databases, the use of flash or solid-state disks (which also applies within transaction processing), clustered versus no-clustered solutions and so on.

6. **Big Data**
To be clear, big data does not necessarily mean lots of data. What it really refers to is the ability to process any type of data: what is typically referred to as semi-structured and unstructured data as well as structured data. Current thinking is that these will typically live alongside conventional solutions as separate technologies, at least in large organisations, but this will not always be the case.

Integrating Trends

Projects involving databases should not be viewed and appreciated solely on how they adhere to these trends. Ideally, each tool or process available should merge in some meaningful way with existing operations. It is important to look of these trends as items that can coincide: enhancing security and moving to the cloud coexist?

Distributed and Deductive Database

Distributed Database:- This chapter introduces the concept of DDBMS. In a distributed database, there are a number of databases that may be geographically distributed all over the world. A distributed DBMS manages the distributed database in a manner so that it appears as one single database to users. In the later part of the chapter, we go on to study the factors that lead to distributed databases, its advantages and disadvantages.

A **distributed database** is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

Features

- Databases in the collection are logically interrelated with each other. Often they represent a single logical database.
- Data is physically stored across multiple sites. Data in each site can be managed by a DBMS independent of the other sites.
- The processors in the sites are connected via a network. They do not have any multiprocessor configuration.
- A distributed database is not a loosely connected file system.
- A distributed database incorporates transaction processing, but it is not synonymous with a transaction processing system.

Distributed Database Management System

A distributed database management system (DDBMS) is a centralized software system that manages a distributed database in a manner as if it were all stored in a single location.

Features

- It is used to create, retrieve, update and delete distributed databases.
- It synchronizes the database periodically and provides access mechanisms by the virtue of which the distribution becomes transparent to the users.
- It ensures that the data modified at any site is universally updated.
- It is used in application areas where large volumes of data are processed and accessed by numerous users simultaneously.
- It is designed for heterogeneous database platforms.

- It maintains confidentiality and data integrity of the databases.

Factors Encouraging DDBMS

The following factors encourage moving over to DDBMS −

- **Distributed Nature of Organizational Units** − Most organizations in the current times are subdivided into multiple units that are physically distributed over the globe. Each unit requires its own set of local data. Thus, the overall database of the organization becomes distributed.
- **Need for Sharing of Data** − The multiple organizational units often need to communicate with each other and share their data and resources. This demands common databases or replicated databases that should be used in a synchronized manner.
- **Support for Both OLTP and OLAP** − Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) work upon diversified systems which may have common data. Distributed database systems aid both these processing by providing synchronized data.
- **Database Recovery** − One of the common techniques used in DDBMS is replication of data across different sites. Replication of data automatically helps in data recovery if database in any site is damaged. Users can access data from other sites while the damaged site is being reconstructed. Thus, database failure may become almost inconspicuous to users.
- **Support for Multiple Application Software** − Most organizations use a variety of application software each with its specific database support. DDBMS provides a uniform functionality for using the same data among different platforms.

Advantages of Distributed Databases

Following are the advantages of distributed databases over centralized databases.

Modular Development − If the system needs to be expanded to new locations or new units, in centralized database systems, the action requires substantial efforts and disruption in the existing functioning. However, in distributed databases, the work simply requires adding new computers and local data to the new site and finally connecting them to the distributed system, with no interruption in current functions.

More Reliable − In case of database failures, the total system of centralized databases comes to a halt. However, in distributed systems, when a component fails, the functioning of the system continues may be at a reduced performance. Hence DDBMS is more reliable.

Better Response − If data is distributed in an efficient manner, then user requests can be met from local data itself, thus providing faster response. On the other hand, in centralized systems, all queries have to pass through the central computer for processing, which increases the response time.

Lower Communication Cost − In distributed database systems, if data is located locally where it is mostly used, then the communication costs for data manipulation can be minimized. This is not feasible in centralized systems.

Adversities of Distributed Databases

Following are some of the adversities associated with distributed databases.

- **Need for complex and expensive software** − DDBMS demands complex and often expensive software to provide data transparency and co-ordination across the several sites.
- **Processing overhead** − Even simple operations may require a large number of communications and additional calculations to provide uniformity in data across the sites.
- **Data integrity** − The need for updating data in multiple sites pose problems of data integrity.
- **Overheads for improper data distribution** − Responsiveness of queries is largely dependent upon proper data distribution. Improper data distribution often leads to very slow response to user requests.

Deductive Database:- A deductive database is a finite collection of facts and rules. By applying the rules of a deductive database to the facts in the database, it is possible to infer additional facts, i.e. facts that are implicitly true but are not explicitly represented in the database.

This paper is a brief introduction to deductive databases. In the first section, we talk about traditional databases, i.e. sets of simple facts. After that, we introduce logic programs, i.e. sets of rules. We then show how to use rules in defining views of a database, in writing constraints on the database, and in defining updates to the database. We close with a brief discussion of special, built-in functions and relations.

2. Databases

When we think about the world, we usually think in terms of objects and relationships among these objects. Objects include things like people and offices and buildings. Relationships include things like the parenthood, ancestry, office assignments, office locations, and so forth.

In sentential databases, we encode each instance of a relationship in the form of a sentence consisting of a relation constant representing the relationship and some terms representing the objects involved in the instance.

The vocabulary of a database is a collection of object constants, function constants, and relation constants. Each function constant and relation constant has an associated arity, i.e. the number of objects involved in any instance of the corresponding function or relation.

A term is either a symbol or a functional term. A functional term is an expression consisting of an n-ary function constant and n terms. In what follows, we write functional terms in traditional mathematical notation - the function followed by
its arguments enclosed in parentheses and separated by commas. For example, if f is a binary function constant and if a and b are object constants,
then f(a,a) and f(a,b) and f(b,a) and f(b,b) are all functional terms. Functional terms can be nested within other functional terms. For example, if f(a,b) is a functional term, then so is f(f(a,b),b).

A datum is an expression formed from an n-ary relation constant and n terms. We write data in mathematical notation. For example, we might write parent(art,bob) to express the fact that Art is the parent of Bob.

A dataset is any set of data that can be formed from the vocabulary of a database. Intuitively, we can think of the data in a dataset as the facts that we believe to be true in the world; data that are not in the dataset are assumed to be false.

As an example of these concepts, consider a small interpersonal database. The objects in this case are people. The relationships specify properties of these people and their interrelationships.

In our example, we use the binary relation constant parent to specify that one person is a parent of another. The sentences below constitute a database describing six instances of the parent relation. The person named art is a parent of the person named bob; art is also a parent of bea, and so forth.

parent(art,bob)

parent(art,bea)

parent(bob,carl)

parent(bea,coe)

parent(carl,daisy)

parent(carl,daniel)

The adult relation is unary relation, i.e. a simple property of a person, not a relationship other people. Everyone in our database is an adult except for daisy and daniel.

adult(art)

adult(bob)

adult(bea)

adult(carl)

adult(coe)

We can express gender with two unary relation constants male and female. The following data expresses the genders of all of the people in our database. Note that, in principle, we need only one relation here, since one gender is the complement of the other. However, representing both allows us to enumerate instances of both gender equally efficiently, which can be useful in certain applications.

male(art)	female(bea)
male(bob)	female(coe)
male(cal)	female(daisy)
male(daniel)	

As an example of a ternary relation, consider the data shown below. Here, we use prefers to represent the fact that the first person likes the second person more than the third person. For example, the first sentence says that Art prefers bea to bob; the second sentence says that carl prefers daisy to daniel.

prefers(art,bea,bob)

prefers(carl,daisy,daniel)

Note that the order of arguments in such sentences is arbitrary. Given the meaning of the prefers relation in our example, the first argument denotes the subject, the second argument is the person who is preferred, and the third argument denotes the person who is less preferred. We could equally well have interpreted the arguments in other orders. The important thing is consistency - once we choose to interpret the arguments in one way, we must stick to that interpretation everywhere.

3. Logic Programs

The rules in a deductive database are often called a logic program. The language of logic programs includes the language of databases but provides additional expressive features.

One key difference is the inclusion of a new type of symbol, called a variable. Variables allow us to state relationships among objects without explicitly naming those objects. In what follows, we use individual capital letters as variables, e.g. X, Y, Z.

In the context of logic programs, a term is defined as an object constant, a variable, or a functional term, i.e. an expression consisting of an n-ary function constant and n simpler terms.

An atom in a logic program is analogous to a datum in a database except that the constituent terms may include variables.

A literal is either an atom or a negation of an atom (i.e. an expression stating that the atom is false). A simple atom is called a positive literal, The negation of an atom is called a negative literal. In what follows, we write negative literals using the negation sign ~. For example, if p(a,b) is an atom, then ~p(a,b) denotes the negation of this atom.

A rule is an expression consisting of a distinguished atom, called the head and a conjunction of zero or more literals, called the body. The literals in the body are called subgoals. In what follows, we write rules as in the example shown below. Here, r(X,Y) is the head, p(X,Y) & ~q(Y) is the body; and p(X,Y) and ~q(Y) are subgoals.

```
r(X,Y) :- p(X,Y) & ~q(Y)
```

Semantically, a rule is something like a reverse implication. It is a statement that the conclusion of the rule is true whenever the conditions are true. For example, the rule above states that r is true of any object X and any object Y if p is true of X and Y and q is not true of Y. For example, if we know p(a,b) and we know that q(b) is false, then, using this rule, we can conclude that r(a,b) must be true.

Exercise: Click here to test your understanding of rule syntax.

A logic program is a finite set of atoms and rules as just defined. In order to simplify our definitions and analysis, we occasionally talk about infinite sets of rules. While these sets are useful, they are not themselves logic programs.

Unfortunately, the language of rules, as defined so far, allows for logic programs with some unpleasant properties. To avoid programs of this sort, it is common in deductive databases to add a couple of restrictions that together eliminate these problems.

The first restriction is safety. A rule in a logic program is safe if and only if every variable that appears in the head or in any negative literal in the body also appears in at least one positive literal in the body. A logic program is safe if and only if every rule in the program is safe.

All of the examples above are safe. By contrast, the two rules shown below are not safe. The first rule is not safe because the variable Z appears in the head but does not appear in any positive subgoal. The second rule is not safe because the variable Z appears in a negative subgoal but not in any positive subgoal.

```
s(X,Y,Z) :- p(X,Y)

t(X,Y) :- p(X,Y) & ~q(Y,Z)
```

To see why safety is matters in the case of the first rule, suppose we had a database in which p(a,b) is true. Then, the body of the first rule is satisfied if we let X be a and Y be b. In this case, we can conclude that every corresponding instance of the head is true. But what should we substitute for Z? Intuitively, we could put anything there; but there could be infinitely many possibilities. For example, we could write any number there. While this is conceptually okay, it is practically problematic.

To see why safety matters in the second rule, suppose we had a database with just two facts, viz. p(a,b) and q(b,c). In this case, if we let X be a and Y be b and Z be anything other than c, then both subgoals true, and we can conclude t(a,b). The main problem with this is that many people incorrectly interpret that negation as meaning there is no Z for which q(Y,Z) is true, whereas the correct reading is that q(Y,Z) needs to be false for just one binding of Z. As we will see in our examples below, there is a simple way of expressing this other meaning without writing unsafe rules.

In logic programming, these problems are avoided by requiring all rules to be safe. While this does restrict what one can say, the good news is that it is usually possible to ensure safety by adding additional subgoals to rules to ensure that the restrictions are satisfied.

Exercise: Click here to test your understanding of the concept of safety.

The second restriction is called stratified negation. It is essential in order to avoid ambiguities. Unfortunately, it is a little more difficult to understand than safety.

The dependency graph for a logic program is a directed graph with two type of arcs, positive and negative. The nodes in the dependency graph for a program represent the relations in the program. There is a positive arc in the graph from one node to another if and only if the former node appears in a positive subgoal of a rule in which the latter node appears in the head. There is a negative arc from one node to another if and only if the former node appears in a negative subgoal of a rule in which the latter node appears in the head.

As an example, consider the following logic program. r(X,Y) is true
if p(X,Y) and q(Y) are true. s(X,Y) is true if r(X,Y) is true and s(Y,X) is false.

```
r(X,Y) :- p(X,Y) & q(Y)

s(X,Y) :- r(X,Y) & ~s(Y,X)
```

The dependency graph for this program contains nodes for p, q, r, and s. Due to the first rule, there is a positive arc from p to r and a positive arc from q to r. Due to the second rule, there is a positive arc from r to s and a negative arc from s to itself.

A negation in a logic program is said to be stratified with respect to negation if and only if there is no negative arc in any cycle in the dependency graph. The logic program just shown is not stratified with respect to negation because there is a cycle involving a negative arc.

The problem with unstratified logic programs is that there is a potential ambiguity. As an example, consider the program above and assume we had a database
containing p(a,b), p(b,a), q(a), and q(b). From these facts we can
conclude r(a,b) and r(b,a) are both true. So far so good. But what can we say about s? If we take s(a,b) to be true and s(b,a) to be false, then the second rule is satisfied. If we take s(a,b) to be false and s(b,a) to be true, then the second rule is again satisfied. We can also take them both to be true. The upshot is that there is ambiguity about s. By concentrating exclusively on programs that are stratified with respect to negation, we avoid such ambiguities.

Exercise: Click here to test your understanding of the concept of stratified negation.

It is common in logic programming to require that all logic programs be both safe and stratified with respect to negation. The restrictions are easy to satisfy in most applications; and, by obeying these restrictions, we ensure that our logic programs produce finite, unambiguous answers for all questions.

4. View Definitions

The principle use of rules is to define new relations in terms of existing relations. The new relations defined in this way are often called view relations (or simply views) to distinguish them from base relations, which are defined by explicit enumeration of instances.

To illustrate the use of rules in defining views, consider once again the world of interpersonal relations. Starting with the base relations, we can define various interesting view relations.

As an example, consider the sentences shown below. The first sentence defines the father relation in terms of parent and male. The second sentence defines mother in terms of parent and female.

```
father(X,Y) :- parent(X,Y) & male(X)

mother(X,Y) :- parent(X,Y) & female(X)
```

The rule below defines the grandparent relation in terms of the parent relation. A person X is the grandparent of a person Z if X is the parent of a person Y and Y is the parent of Z. The variable Y here is a thread variable that connects the first subgoal to the second but does not itself appear in the head of the rule.

```
grandparent(X,Z) :- parent(X,Y) & parent(Y,Z)
```

Note that the same relation can appear in the head of more than one rule. For example, the person relation is true of a person Y if there is an X such that X is the parent of Y or if Y is the parent of some person Z. Note that in this case the conditions are disjunctive (at least one must be true), whereas the conditions in the grandfather case are conjunctive (both must be true).

```
person(X) :- parent(X,Y)

person(Y) :- parent(X,Y)
```

A person X is an ancestor of a person Z if X is the parent of Z or if there is a person Y such that X is an ancestor of and Y is an ancestor of Z. This example shows that is possible for a relation to appear in its own definition. (But recall our discussion of stratification for a restriction on this capability.)

```
ancestor(X,Y) :- parent(X,Y)

ancestor(X,Z) :- ancestor(X,Y) & ancestor(Y,Z)
```

A childless person is one who has no children. We can define the property of being childless with the rules shown below. The first rule states that a person X is childless if X is a person and it is not the case that X is a parent. The second rule says that isparent is true of X if X is the parent of some person Y.

```
childless(X) :- person(X) & ~isparent(X,Y)

isparent(X) :- parent(X,Y)
```

Note the use of the helper relation isparent here. It is tempting to write the childless rule as childless(X) :- person(X) & ~parent(X,Y). However, this would be wrong. This would define X to be childless if X is a person and there is some Y such that X is ~parent(X,Y) is true. But we really want to say that ~parent(X,Y) holds for all Y. Defining isparent and using its negation in the definition of childless allows us to express this universal quantification.

5. Errors and Warnings

In our development thus far, we have assumed that the extension of an n-ary relation may be any set of n-tuples from the domain. This is rarely the case. Often, there are constraints that limit the set of possibilities. For example, a person cannot be his own parent. In some cases, constraints involve multiple relations. For example, all parents are adults; in other words, if an entity appears in the first column of the parent relation, it must also appear as an entry in the adult relation.

In many database texts, constraints are written in direct form - by writing rules that say, in effect, that if certain things are true in an extension, then other things must also be true. The inclusion dependency mentioned above is an example - if an entity appears in the first column of the parent relation, it must also appear as an entry in the adult relation.

In what follows, we use a slightly less direct approach - we encode limitations by writing rules that say when a database is not well-formed. We simply invent a new 0-ary relation, here called illegal, and define it to be true in any extension that does not satisfy our constraints.

This approach works particularly well for consistency constraints like the one stating that a person cannot be his own parent.

```
illegal :- parent(X,X)
```

It also works well for mutual exclusion constraints like the one below, which states that a person cannot be in both the male and the female relations.

```
illegal :- male(X) & female(X)
```

Using this technique, we can also write the inclusion dependency mentioned earlier. There is an error if an entity is in the first column of the parent relation and it does not occur in the adult relation.

illegal :- parent(X,Y) & ~adult(X)

Database management systems can use such constraints in a variety of ways. They can be used to optimize the processing of queries. They can also be used to check that updates do not lead to unacceptable extensions.

6. Updates

In updating a database, a user specifies a sentence to add to a database or a sentences to delete. In some cases, the user can group several changes of this sort in a single, so-called, atomic transaction. If the result of executing the transaction satisfies the constraints, the update is performed; otherwise it is rejected.

Unfortunately, if a user forgets to include an addition or deletion required by the constraints, this can lead to errors. In order to simplify the update process for the user, some database systems provide the administrator the ability to write update rules, i.e. rules that are executed by the system to augment a specified transaction with the additions and deletions necessary to avoid errors. In what follows, we show one way that this can be done

Our update language includes four special operators - pluss, minus, pos, and neg. pluss takes a sentence as argument and is true if and only if the user specifies that sentence as an addition in a transaction. minus takes a sentence as argument and is true if and only if the user specifies that sentence as an addition in a transaction. pos takes a sentence as argument and is true if and only if the system concludes that the specified sentence should be added to the database. neg takes a sentence as argument and is true if and only if the system concludes that the specified sentence should be added to the database. Update rules are rules that define pos and neg in terms of pluss and minus and the current state of the database.

As an example of this mechanism in action, consider the rules shown below. The first dictates that the system remove a sentence of the form male(X) whenever the user adds a sentence of the form female(X). The second rule is analogous to the first with male and female reversed. Together, these two rules enforce the mutual exclusion on male and female.

neg(male(X)) :- pluss(female(X))

neg(female(X)) :- pluss(male(X))

Similarly, we can enforce the inclusion dependency on parent and adult by writing the following rule. If the user adds a sentence of the form parent(X,Y), then the system also adds a sentence of the form adult(X).

```
pos(adult(X)) :- pluss(parent(X,Y))
```

Another use of this update mechanism is to maintain materialized views. (A materialized view is a defined relation that is stored explicitly in the database, usually to save recomputation.)

Suppose, for example, we were to materialize the father relation defined earlier. Then we could write the update rules to maintain this materialized view. According to the first rule, the system should add a sentence of the form father(X,Y) whenever the user adds parent(X,Y) and male(X) is know to be true and the user does not delete that fact. The other rules cover the other cases.

```
pos(father(X,Y)) :- pluss(parent(X,Y)) & male(X) & ~minus(male(X))

pos(father(X,Y)) :- parent(X,Y) & pluss(male(X)) & ~minus(parent(X,Y))

pos(father(X,Y)) :- pluss(parent(X,Y)) & pluss(male(X))

neg(father(X,Y)) :- minus(parent(X,Y))

neg(father(X,Y)) :- minus(male(X))
```

Note that not all constraints can be enforced using update rules. For example, if a user suggests adding the sentence parent(art,art) to the database in our interpersonal relations example, there is nothing the system can do to repair this error except to reject the transaction. In some cases, there is no way to make a repair unambiguously; more information is needed from the user. For example, we might have a constraint that every person is in either the male or the female relation. If the user specifies a parent fact involving a new person but does not specify the gender of that person, there is no way for the system to decide that gender for itself.

7. Special Relations

In practical logic programming languages, it is common to "build in" commonly used concepts. These typically include arithmetic functions (such as +, *, max, min), string functions (such as concatenation), comparison operators (such as < and >), and equality (=). It is also common to include aggregate operators, such as countofall, avgofall sumofall, and so forth.

In many practical logic programming languages, mathematical functions are represented as relations. For example, the the binary addition operator + is often represented by the the ternary relation constant plus. For example, the following rule defines the combined age of two people. The combined age of X and Y is S if the age of X is M and the age of Y is N and S is the result of adding M andN.

```
combinedage(X,Y,S) :- age(X,M) & age(Y,N) & plus(M,N,S)
```

Similarly, aggregate operators are typically represented as relations. For example the following rule defines the number of a person's grandchildren using
the countofall relation in this way. N is the number of grandchildren of X if N is the count of all Z such that X is the grandparent of Z.

```
grandchildren(X,N) :- person(X) & countofall(Z,grandparent(X,Z),N)
```

In logic programming languages that provide such built-in concepts, there are usually syntactic restrictions on their use. For example, if a rule contains a subgoal with a comparison relation, then every variable that occurs in that subgoal must occur in at least one positive literal in the body and that occurrence must precede the subgoal with the comparison relation. If a rule mentions an arithmetic function, then any variable that occurs in all but the last position of that subgoal must occur in at least one positive literal in the body and that occurrence must precede the subgoal with the arithmetic relation.