

Block-1

Unit-1

- 1.1** Learning Objectives
- 1.2** Introduction
- 1.3** Database system and application
- 1.4** Purpose of database system
- 1.5** Characteristics and Benefits of a Database
- 1.6** Components of DBMS
- 1.7** Merits and Demerits of DBMS
- 1.8** Database Architecture
- 1.9** Traditional file systems
- 1.10** View of data
- 1.11** Database language languages
- 1.12** Data Dictionary
- 1.13** Types of DBMS
 - 1.13.1** Centralized DBMS
 - 1.13.2** Parallel DBMS
 - 1.13.3** Distributed DBMS
 - 1.13.4** Client-Server DBMS
- 1.14** Relational databases
- 1.15** Database Design
- 1.16** Database Administrator
- 1.17** Check Your Progress
- 1.18** Answer to Check Your Progress

1.1 Learning Objectives

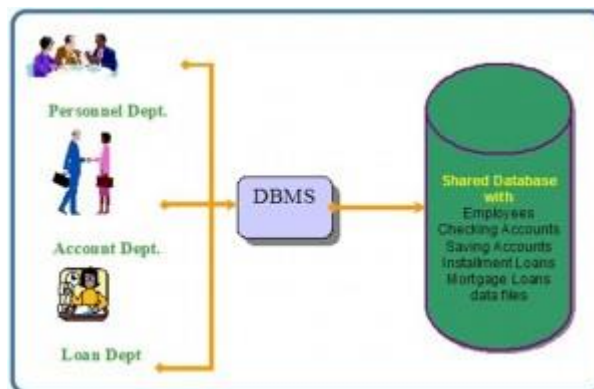
After going through this unit, the learner will be able to learn:

- The Database system and application
- The Purpose of the database system
- About Characteristics and Benefits of a Database
- About Database Architecture
- About Database language languages
- About the Data Dictionary
- Types of DBMS
- The Relational databases
- About Database Design
- The role of the database administrator

1.2 Introduction

A *database management system (DBMS)* is a collection of programs that enables users to create and maintain databases and control all access to them. The primary goal of a DBMS is to provide an environment that is both convenient and efficient for users to retrieve and store information.

With the database approach, we can have the traditional banking system as shown in Figure 2.3. In this bank example, a DBMS is used by the Personnel Department, the Account Department and the Loan Department to access the shared corporate database.



1.3 Database system and Application

A **database application** is a computer program whose primary purpose is entering and retrieving information from a computerized database. Early examples of database applications were accounting systems and airline reservations systems, such as SABRE, developed starting in 1957.

A characteristic of modern database applications is that they facilitate simultaneous updates and queries from multiple users. Systems in the 1970s might have accomplished this by having each user in front of a 3270 terminal to a mainframe computer. By the mid-1980s it was becoming more common to give each user a personal computer and have a program running on that PC that connected to a database server. Information would be pulled from the database, transmitted over a network, and then arranged, graphed, or otherwise formatted by the program running on the PC. Starting in the mid-1990s it became more common to build database applications with a Web interface. Rather than develop custom software to run on a user's PC, the user would use the same Web browser program for every application. A database application with a Web interface had the advantage that it could be used on devices of different sizes, with different hardware, and with different operating systems. Examples of early database applications with Web interfaces include amazon.com, which used the Oracle relational database management system, the photo.net online community, whose implementation on top of Oracle was described in the book *Database-Backed Web Sites* (Ziff-Davis Press; May 1997), and eBay, also running Oracle.¹

Electronic medical records are referred to on emrexperts.com, in December 2010, as "a software database application". A 2005 O'Reilly book uses the term in its title: *Database Applications and the Web*.

Some of the most complex database applications remain accounting systems, such as SAP, which may contain thousands of tables in only a single module. Many of today's most widely used computer systems are database applications, for example, Facebook, which was built on top of MySQL.

The etymology of the phrase "database application" comes from the practice of dividing computer software into systems programs, such as the operating system, compilers, the file system, and tools such as the database management system, and application programs, such as a payroll check processor. On a standard PC running Microsoft Windows, for example, the Windows operating system contains all of the systems programs while games, word processors, spreadsheet programs, photo editing programs, etc. would be application

programs. As "application" is short for "application program", "database application" is short for "database application program".

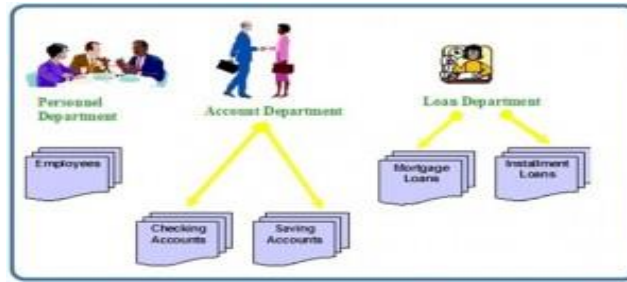
Not every program that uses a database would typically be considered a "database application". For example, many physics experiments, e.g., the Large Hadron Collider, generate massive data sets that programs subsequently analyze. The data sets constitute a "database", though they are not typically managed with a standard relational database management system. The computer programs that analyze the data are primarily developed to answer hypotheses, not to put information back into the database and therefore the overall program would not be called a "database application".

1.4 Purpose of Database System

A database is collection of interrelated data organized in such a way so that it is easy to access, retrieve, manage and understand data. Say, for example, a list of students - you will have a list of the names, along with their roll nos. You may also have a table somewhere about the marks of each student on your list. There will be separate files for mark lists for different subjects, and courses. These mark lists may be queried to find the aggregate marks, and overall percentages, which may be stored in some result files. All this collective information, derived from raw data, is called a database.

One way to keep information on a computer is to store it in permanent files. A company system has a number of application programs; each of them is designed to manipulate data files. These application programs have been written at the request of the users in the organization. New applications are added to the system as the need arises. The system just described is called the *file-based system*.

Consider a traditional banking system that uses the file-based system to manage the organization's the data in the following figure As we can see, there are different departments in the bank. Each has its own applications that manage and manipulate different data files. For banking systems, the programs may be used to debit or credit an account, find the balance of an account, add a new mortgage loan and generate monthly statements.



1.5 Characteristics and Benefits of a Database

There are a number of characteristics that distinguish the database approach from the file-based system or approach. This unit describes the benefits (and features) of the database system.

Self-describing nature of a database system

A database system is referred to as self-describing because it not only contains the database itself, but also metadata which defines and describes the data and relationships between tables in the database. This information is used by the DBMS software or database users if needed. This separation of data and information about the data makes a database system totally different from the traditional file-based system in which the data definition is part of the application programs.

Insulation between program and data

In the file-based system, the structure of the data files is defined in the application programs so if a user wants to change the structure of a file, all the programs that access that file might need to be changed as well.

On the other hand, in the database approach, the data structure is stored in the system catalogue and not in the programs. Therefore, one change is all that is needed to change the structure of a file. This insulation between the programs and data is also called program-data independence.

Support for multiple views of data

A database supports multiple views of data. A view is a subset of the database, which is defined and dedicated for particular users of the system. Multiple users in the system might have different views of the system. Each view might contain only the data of interest to a user or group of users.

Sharing of data and multiuser system

Current database systems are designed for multiple users. That is, they allow many users to access the same database at the same time. This access is achieved through features called concurrency control strategies. These strategies ensure that the data accessed are always correct and that data integrity is maintained.

The design of modern multiuser database systems is a great improvement from those in the past which restricted usage to one person at a time.

Control of data redundancy

In the database approach, ideally, each data item is stored in only one place in the database. In some cases, data redundancy still exists to improve system performance, but such redundancy is controlled by application programming and kept to minimum by introducing as little redundancy as possible when designing the database.

Data sharing

The integration of all the data, for an organization, within a database system has many advantages. First, it allows for data sharing among employees and others who have access to the system. Second, it gives users the ability to generate more information from a given amount of data than would be possible without the integration.

Enforcement of integrity constraints

Database management systems must provide the ability to define and enforce certain constraints to ensure that users enter valid information and maintain data integrity. A database constraint is a restriction or rule that dictates what can be entered or edited in a table such as a postal code using a certain format or adding a valid city in the City field.

There are many types of database constraints. Data type, for example, determines the sort of data permitted in a field, for example numbers only. Data uniqueness such as the primary key ensures that no duplicates are entered. Constraints can be simple (field based) or complex (programming).

Restriction of unauthorized access

Not all users of a database system will have the same accessing privileges. For example, one user might have read-only access (i.e., the ability to read a file but not make changes), while another might have read and write privileges, which is the ability to both read and modify a file. For this reason, a database management system should provide a security subsystem to create and control different types of user accounts and restrict unauthorized access.

Data independence

Another advantage of a database management system is how it allows for data independence. In other words, the system data descriptions or data describing data (metadata) are separated from the application programs. This is possible because changes to the data structure are handled by the database management system and are not embedded in the program itself.

Transaction processing

A database management system must include concurrency control subsystems. This feature ensures that data remains consistent and valid during transaction processing even if several users update the same information.

Provision for multiple views of data

By its very nature, a DBMS permits many users to have access to its database either individually or simultaneously. It is not important for users to be aware of how and where the data they access is stored.

Backup and recovery facilities

Backup and recovery are methods that allow you to protect your data from loss. The database system provides a separate process, from that of a network backup, for backing up and recovering data. If a hard drive fails and the database stored on the hard drive is not accessible, the only way to recover the database is from a backup.

If a computer system fails in the middle of a complex update process, the recovery subsystem is responsible for making sure that the database is restored to its original state. These are two more benefits of a database management system.

1.6 Components of DBMS

A database management system (DBMS) consists of several components. Each component plays very important role in the database management system environment. The major components of database management system are:

- Software
- Hardware
- Data
- Procedures
- Users
- **HARDWARE:** The hardware is the actual computer system used for keeping and accessing the database. It consists of a set of physical electronic devices such as

computers (together with associated I/O devices like disk drives), storage devices, I/O channels, electromechanical devices that make interface between computers and the real world systems etc, and so on. It is impossible to implement the DBMS without the hardware devices, In a network, a powerful computer with high data processing speed and a storage device with large storage capacity is required as database server.

- **SOFTWARE:** The main component of a DBMS is the software. It is the set of programs used to handle the database and to control and manage the overall computerized database. DBMS software itself, is the most important software component in the overall system.
- **DATA:** Data is the most important component of the DBMS. The main purpose of DBMS is to process the data. In DBMS, databases are defined, constructed and then data is stored, updated and retrieved to and from the databases. The database contains both the actual data and the metadata.
- **PROCEDURES:** Procedures refer to the instructions and rules that help to design the database and to use the DBMS. The users that operate and manage the DBMS require documented procedures on how to use or run the database management system.
- **USERS:** The users are the people who manage the databases and perform different operations on the databases in the database system. There are three kinds of people who play different roles in database system
- **Application Programmers:** These users implement specific application programs to access the stored data. They must be familiar with the DBMSs to accomplish their task.
- **Database Administrators:** This may be one person or a group of people in an organization responsible for authorizing access to the database, monitoring its use and managing all of the resources to support the use of the entire database system.
- **End-Users:** End users are the people whose jobs require access to a database for querying, updating and generating reports.

1.7 Merits and Demerits of DBMS

Merits of DBMS

1. Improved data sharing.

The DBMS helps create an environment in which end users have better access to more and better-managed data.

2. Improved data security.

The more the users access the data, the greater the risks of data security breaches. This is the reason DBMS provides a framework for better enforcement of data privacy and security policies.

3. Better data integration.

It is much easier to see how actions in one segment of the company affect other segments.

4. Minimized data inconsistency.

Data inconsistency exists when different versions of the same data appear in different places. The probability of data inconsistency is greatly reduced in a properly designed database.

5. Improved data access.

The DBMS makes it possible to produce quick answers to ad hoc queries. From a database perspective, a query is a specific request issued to the DBMS for data manipulation.

6. Improved decision making.

Better-managed data and improved data access make it possible to generate better-quality information, on which better decisions are based. Data quality is a comprehensive approach to promoting the accuracy, validity, and timeliness of the data. While the DBMS does not guarantee data quality, it provides a framework to facilitate data quality initiatives.

7. Increased end-user productivity.

The availability of data, combined with the tools that transform data into usable information, empowers end users to make quick, informed decisions that can make the difference between success and failure in the global economy.

Demerits of DBMS

1. Complexity:

The provision of the functionality that is expected of a good DBMS makes the DBMS an extremely complex piece of software. 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:

The DBMS file based system 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 availability 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:

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 of 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 is one of the main reasons why some organizations feel tied to their current systems and cannot switch to modern database technology.

1.8 Database Architecture

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

There are following three levels or layers of DBMS architecture:

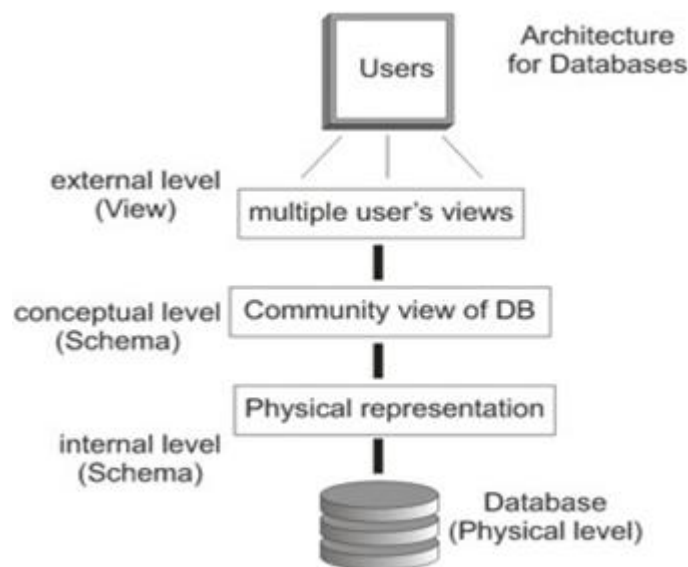
1. External Level
2. Conceptual Level
3. Internal Level

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

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

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

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



Three levels or layers of DBMS architecture

1.9 Traditional file systems

Before the advent of modern general purpose database systems, traditional file systems were used to store, manipulate, retrieve and delete data. These file systems comprised of two major components: the data stored as a collection of files, and several application programs that accessed and manipulated the files. These systems were very complicated because of a variety of reasons. Data was not centralized, meaning there could be two or more copies of same data, resulting in redundancy. Redundancy would lead to difficulty in maintaining consistency throughout the system. Often, application programs

were targeted to achieve some of the major queries, such as adding new tuples, removing some data, manipulating existing data, etc. However, for each new query, either a new application had to be written, or some entirely different programs were to be clubbed together in a complex manner to achieve the desired result.

As technology improved, more and more advances were made in the existing database systems. Progress was made towards more efficient systems, more application programs were added to generalize individual systems, among other changes. This resulted in different types of database systems, as follows:

1. **Single file or flat file database system:** This is the traditional file based data storage system, consisting of collection of files and related application programs.
2. **Hierarchical system:** Data is stored in a hierarchical format, with parent entities at the top, linked to its child entities. As the name suggests, hierarchical system enforces a tree like structure, with the only links possible between parent and its children. There is no link between siblings, which may or may not be a disadvantage, based on the user requirements. The implementation is pretty straightforward, and hence it is highly recommended for storing and managing structured data. An example approach is using XML (eXtensible Markup Language).
3. **Networked system:** A network system allows for more interconnections between data, that is when many-to-many relationships exist between records, network systems should be used. It is more efficient and useful for large scale projects, such as maintaining company records, where records may not necessarily be stored in a hierarchy. Network based DBMS are useful for managing structured data. High level languages such as Pascal, Cobol, C++ are used to implement network – based database systems.
4. **Relational database system:** This type of system is one of the most popular systems in use today. It emphasizes the importance of relations between records and entities. This paves way for more flexibility and more extensibility. Data is stored in tabular form, and relationships are clearly defined between each entity or table. Such DBMS are also called as RDBMS. Several popular RDBMS are Oracle, MS Access, SQL.
5. **Object-oriented approach:** This is also one of the very popular systems, and the most modern one. Classes and objects are used to describe real world entities. It has a high significance in today's world, where unstructured and semi – structured data have become mainstream. Object-oriented DBMS are comprised of objects and their behavior. Objects

are implemented with the help of a variety of data structures, and this ability to store information of all types and in all formats is what makes this approach attractive. Behavior of objects is implemented with the help of sub – programs called methods, or subroutines, functions etc. One major disadvantage is the cost, as these systems can be expensive and are suitable for large scale projects. These can be implemented using high level languages such as Java, C++.

Some examples of general purpose database systems are MySQL, MS Access, SQL. It is possible for programmers to build their own database system for specific requirements.

1.10 Data View

Database is collection of data or interrelated data contains information about one or more related enterprise or Database is a collection of data, typically describing the activities of one or more related organization. A major purpose of database system is to provide users with an abstract view of the data. That means the system hides certain details of how the data are stored and maintained.

Data Abstraction

To hide certain details of how the data are stored and maintained according to users. There are three types of Data abstraction

- i. Physical level
- ii. Physical level
- iii. View (External) level

Physical Level: The lowest level of abstraction describes how the data are actually stored. It describes complex low-level data structures in detail.

Logical Level: The next higher level of abstraction describes what data are stored in the database, and what relationship exists among those data. At this level the database is described logically in terms of simple data-structure.

View (External) level: It shows only a part of the database relevant to the users.

Instances and Schemas

Instances: The collection of information stored in the database at a particular moment is called an Instances.

Examples: An organization with an employee's database might have three different instances: production (used to contain live data), pre-production (used to test new functionality prior to

release into production) and development (used by database developers to create new functionality).

Schema: The overall design of the database is called the database schema. Schemas are changed infrequently.

Example: Students(sid: string, name: string, login: string, age: integer, gpa: real)

There are three types of Schema

- Conceptual schema
- Physical schema
- External schema

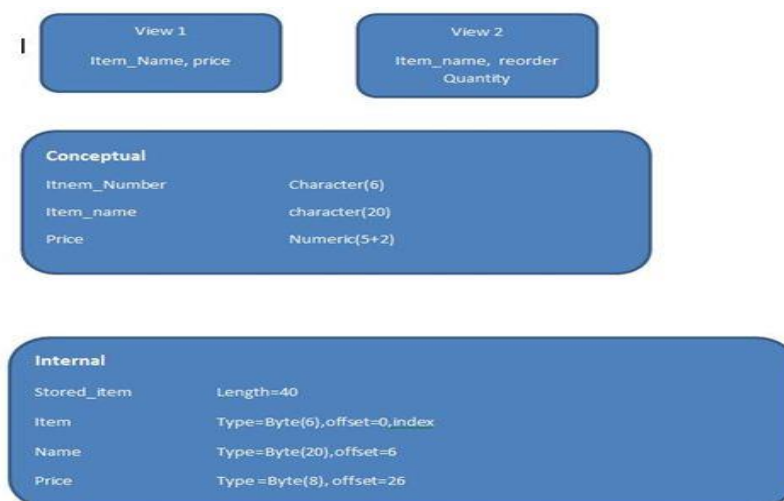
Conceptual Schema (or logical schema): describes the stored data in terms of the data model of the DBMS. In relational DBMS the conceptual schema describes all relations that are stored in the database.

Example– Students(sid: string, name: string, login: string, age: integer, gpa: real)

• **Physical schema (or internal schema):** Specifies additional storage details such as the file organizations used and the auxiliary data structures used for fast retrieval.

• **External Schema:** This level is closest to the user and is concerned with the way in which the data are viewed by individual users.

Example- Info(name: String, login: string)



Data Independence

The ability to modify a schema definition in one level without affecting a scheme definition in the next higher level is called Data Independence. There are two levels of data independence.

- i. **Physical Data Independence:** The ability to modify the scheme followed at the physical level without affecting the scheme at the conceptual level.
- ii. **Logical Data Independence:** The ability to modify the conceptual scheme without causing any changes in the scheme followed at the view levels. i.e. The application program remain same even after changes in conceptual level.

Data Models

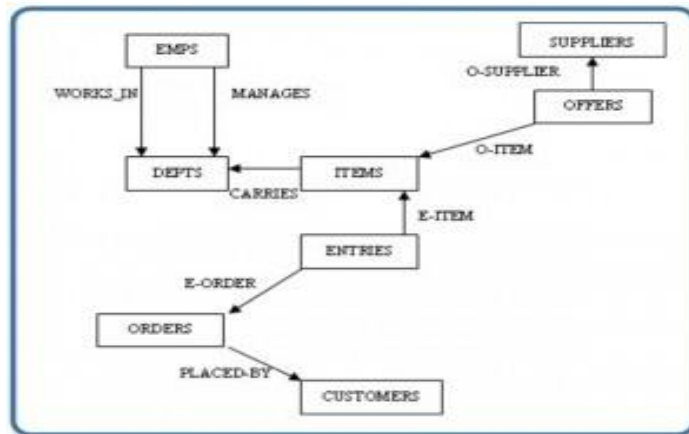
High-level Conceptual Data Models

High-level conceptual data models provide concepts for presenting data in ways that are close to the way people perceive data. A typical example is the entity relationship model, which uses main concepts like entities, attributes and relationships. An entity represents a real-world object such as an employee or a project. The entity has attributes that represent properties such as an employee's name, address and birthdate. A relationship represents an association among entities; for example, an employee works on many projects. A relationship exists between the employee and each project.

Record-based Logical Data Models

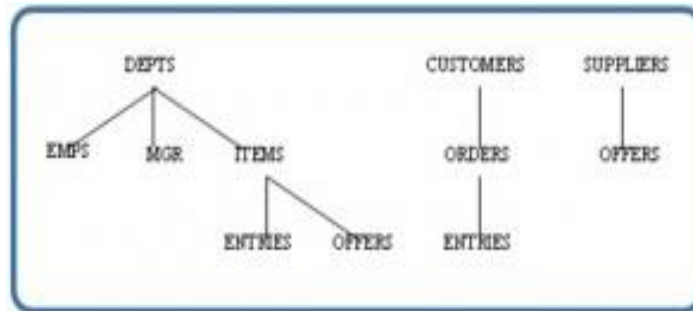
Record-based logical data models provide concepts users can understand but are not too far from the way data is stored in the computer. Three well-known data models of this type are relational data models, network data models and hierarchical data models.

- The relational model represents data as relations, or tables. For example, in the membership system at Science World, each membership has many members. The membership identifier, expiry date and address information are fields in the membership. The members are individuals such as Mickey, Minnie, Mighty, Door, Tom, King, Man and Moose. Each record is said to be an instance of the membership table.
- The network model represents data as record types. This model also represents a limited type of one to many relationship called a set type, as shown in the below Figure



Network model diagram

- The hierarchical model represents data as a hierarchical tree structure. Each branch of the hierarchy represents a number of related records. The below Figure shows this schema in hierarchical model notation.



Hierarchical model diagram.

1.11 Database language languages

A database system provides data- definition language to specify the database schema and A DDL is a language used to define data structures within a database. Data-manipulation language to express database queries and updates. DML is responsible for retrieval, insertion, deletion, modification of information in the database.

Data- Manipulation Language

A data manipulation language (DML) is a family of syntax elements similar to a computer programming language used for selecting, inserting, deleting and updating data in a database. Performing read-only queries of data is sometimes also considered a component of DML. A data manipulation language is a language that enables the database user to access and manipulate the data as organized by the appropriate data model. Data Manipulation Language (DML) statements which are used for managing data within schema objects.

- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - deletes all records from a table, the space for the records remain

Data-Definition Language

A DDL is a language used to define data structures within a database. Basic idea is Hide implementation details of the database schemes from the users. DDL statements are compiled, resulting in a set of tables stored in a special file called a data dictionary or data directory. The data directory contains metadata (data about data). It is typically considered to be a subset of SQL, but can also refer to languages that define other types of data. The DDL concept and name was first introduced in relation to the Codasyl database model, where the schema of the database was written in a language syntax describing the records, fields, and sets of the user data model.

Here are some examples of Data Definition Language (DDL) statements which are used to define the database structure or schema.

- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database

1.12 Data Dictionary

A data dictionary is a centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format. A data dictionary is used in the development of a relational database system.

A data dictionary stores metadata that defines and describes data so that it can be easily understood by anyone who would like to use it or analyze it at a later date. Librarians can assist in building data dictionaries for researchers by working closely with them at the onset of a new research project. The data dictionary – if started at the beginning of a research lifecycle – can assist in the management of research data from the beginning of data capture to the completion of a research study. For example, a data dictionary would be used to define each descriptive heading used when gathering quantitative data as part of a research study.

1.13 Types of DBMS

There are four types of DBMS

- i. Centralized DBMS
 - ii. Parallel DBMS
 - iii. Distributed DBMS
 - iv. Client-Server DBMS
-
- i. Centralized DBMS:** A centralized database (sometimes abbreviated CDB) is a database that is located, stored, and maintained in a single location. This location is most often a central computer or database system, for example a desktop or server CPU, or a mainframe computer. In most cases, a centralized database would be used by an organization (e.g. a business company) or an institution (e.g. a university.) Users access a centralized database through a computer network which is able to give them access to the central CPU, which in turn maintains to the database itself.

 - ii. Parallel DBMS:** A parallel database system seeks to improve performance through parallelization of various operations, such as loading data, building indexes and evaluating queries. Although data may be stored in a distributed fashion, the distribution is governed solely by performance considerations. Parallel databases improve processing and input/output speeds by using multiple CPUs and disks in parallel. Centralized and client–server database systems are not powerful enough to handle such applications. In parallel processing, many operations are performed simultaneously, as opposed to serial processing, in which the computational steps are performed sequentially.

 - iii. Distributed DBMS:** A distributed database (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network. This resource distribution improves performance, reliability, availability and modularity that are inherent in distributed systems. As with traditional centralized databases, distributed database systems (DDBS) must provide an efficient user interface that hides all of the underlying data distribution details of the DDB from the users. The use of a relational query allows the user to specify a description of the data that is required without having to know where the data is physically located.

iv. Client-Server DBMS: A Client-Server Database System consists of three primary software components (aside from the network software and operating systems of the computers in question): the *client application* (also called the front end), the *data access layer* (also called middleware), and the *database server* (also called a database engine, DBMS, data source, or back end).

The client application is responsible for accepting input from the user, submitting a *query* to the database server based on that input, receiving results from the server, formatting them, and presenting them to the user.

The data access layer is relatively transparent to the user, but may be very apparent to the developer of the client app. It provides for the app an API used to submit queries to a data source without much concern for the network between them.

The database server accepts queries from clients, processes them concurrently, and returns results. There are a number of different query languages around, by far the most prevalent of which is SQL. (By the way, contrary to conventional wisdom, “SQL” doesn’t stand for anything in particular. The ‘S’ isn’t for “standard” or “structured,” although the ‘QL’ is thought to stand for “query language.”)

1.14 Relational databases

A relational database is a system for storing and accessing data organized into relations. A relation is a bag of tuples. Each tuple is an ordered sequence of attributes. Each attribute is a data value belonging to some data type. All of the tuples in a relation have the same number of attributes. In addition, the relation has a schema that is imposed on each tuple in the relation, specifying what the data type each attribute in each tuple will have. For example, the relation’s schema might specify that the first attribute in each tuple is an integer. The relation’s schema also gives a name to each attribute. The attribute names give us a convenient way of referring to tuple attributes without having to say “the first attribute”, “the fourth attribute”, etc.

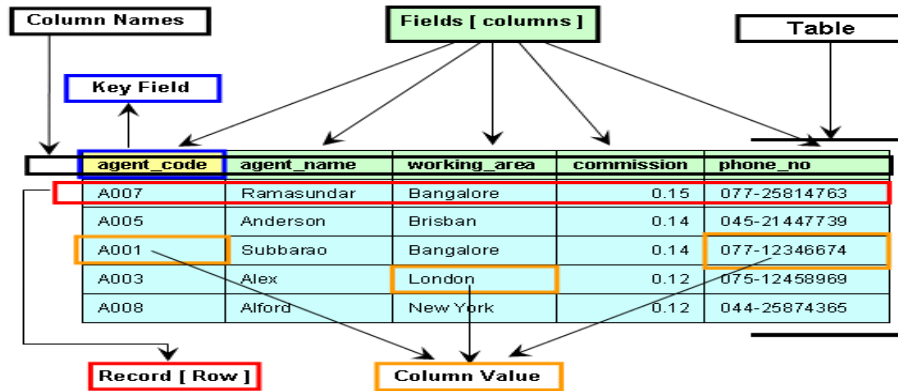
Typical relational databases support numeric and text data types as tuple attributes. Many relational databases also support “BLOBs” (Binary Large Objects) as attributes. A BLOB is a large, uninterpreted chunk of data. BLOBs are useful for storing files, images, and other large chunks of data in a relational database.

Tables

A table is a collection of rows having one or more columns. A row is an instance of a row type. Every row of the same table has the same row type. The value of the n-th field of every row in a table is the value of the n-th column of that row in the table. The row is the smallest unit of data that can be inserted into a table and deleted from a table.

The number of rows in a table is its cardinality. A table whose cardinality is 0 (zero) is said to be empty.

In the following pictorial presentation of a table and different components of it :



A column name can be used in more than one tables and to maintain the integrity of data and reduce redundancy. This is called a relation.

Elements of a table

Fields [columns]

The information of a table stored in some heads, those are fields or columns. Columns show vertically in a table.

Column Names

Each field or column has an individual name. A table cannot contain the same name of two different columns

Record [Row]

All the columns in a table make a row. Each row contains all the information of individual topics.

Column Value

The value of each field makes a row is the column value.

Key Field

Each table should contain a field which can create a link with another one or more table is the key field of a table.

In this relation, there are four attributes called `author_lastname`, `author_firstname`, `title`, and `ISBN`. Each attribute is a text string.

Databases will typically have many relations. One motivation for allowing multiple relations in a database is to avoid storing redundant information. For example, in the relation above, there are two tuples representing books by the same author, Callus Tacticus. Because the author name is represented twice, there is the possibility that this information might not be recorded consistently if the relation were modified.

We can avoid this redundancy by splitting the database into two relations, `books` and `authors`:

The `books` relation:

<code>author_id</code>	<code>title</code>	<code>ISBN</code>
1	A History of Hats	0-651-65165-4
2	Guide to Impossible Buildings	82-234-5475-0
3	First Flights in Witchcraft	5-9672-6521-X
4	Habits of the Wolves	91-33-65168-X
5	Sieges and Survival	0-651-65165-4
5	VENI VIDI VICI: A Soldier's Life	84-15978-99-5

The `authors` relation:

<code>author_id</code>	<code>author_lastname</code>	<code>author_firstname</code>
1	Smallfinger	F.G.
2	Whittlbey	W.H.J.
3	Earwig	Lettice
4	Lightly	W.E.
5	Tacticus	Callus

The `books` relation has been changed so that the author of each book is represented by a unique integer identifier, the `author_id` attribute. This attribute also exists in the `authors` relation. So, the author of each book tuple in the `books` relation is represented indirectly, by reference to a matching author tuple in the `authors` relation.

1.15 Database Design

Database design is the process of producing a detailed data model of database. This data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a data definition language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity. The term database design can be used to describe many different parts of the design of an overall database system. Principally, and most correctly, it can be thought of as the logical design of the base data structures used to store the data. In the relational model these are the tables and views. In an object database the entities and relationships map directly to object classes and named relationships. However, the term database design could also be used to apply to the overall process of designing, not just the base data structures, but also the forms and queries used as part of the overall database application within the database management system (DBMS).

The process of doing database design generally consists of a number of steps which will be carried out by the database designer. Usually, the designer must:

- Determine the data to be stored in the database.
- Determine the relationships between the different data elements.
- Superimpose a logical structure upon the data on the basis of these relationships.

Within the relational model the final step above can generally be broken down into two further steps, that of determining the grouping of information within the system, generally determining what are the basic objects about which information is being stored, and then determining the relationships between these groups of information, or objects. This step is not necessary with an Object database.

1.16 Database Administrator

Database administration is the function of managing and maintaining database management systems (DBMS) software. Mainstream DBMS software such as Oracle, IBM DB2 and Microsoft SQL Server need ongoing management. As such, corporations that use DBMS software often hire specialized IT (Information Technology) personnel called Database Administrators or DBAs.

DBA Responsibilities

- Installation, configuration and upgrading of Database server software and related products.
- Evaluate Database features and Database related products.
- Establish and maintain sound backup and recovery policies and procedures.
- Take care of the Database design and implementation.
- Implement and maintain database security (create and maintain users and roles, assign privileges).
- Database tuning and performance monitoring.
- Application tuning and performance monitoring.
- Setup and maintain documentation and standards.
- Plan growth and changes (capacity planning).
- Work as part of a team and provide 24x7 support when required.
- Do general technical troubleshooting and give cons.
- Database recovery.

1.17 Check Your Progress

1. Select the correct answer:

(a) Which is not a DBMS packages?

- (i) Unify
- (ii) Ingress
- (iii) IDMS
- (iv) All are DBMS packages

(b) Find the wrong statement

Database software

- (i) Provides facilities to create, use and maintain database.
- (ii) Supports report generation, statistical output, graphical output.
- (iii) Provides routine for backup and recovery.
- (iv) All are correct.

(c) Which one of the following is not a valid relational database?

- (i) SYBASE
- (ii) IMS

(iii) ORACLE

(iv) UNIFY

(d) Centralized control is

(i) advantage of a DBMS

(ii) disadvantage of a DBMS

(iii) Both (i) and (ii)

(iv) None of the above

(e) Data are

(i) Raw facts and figures

(ii) Information

(iii) Electronic representation of facts

(iv) None of these

2. Select TRUE or FALSE in the following statements:

(i) The conceptual view is a view of the total database content.

(ii) User's view is also called external view.

(iii) The database schema and an instance of the database are the same thing.

(iv) A view of a database that appears to an application program is known as schema.

(v) Logical data independence indicates that the conceptual schema can be changed without affecting the existing external schemes.

(vi) A database is a computer-based record keeping system whose over all purpose is to record and maintain information.

3. Multiple Choice

(a) A view of database that appear to an application program is known as-----

-

(i) schema

(ii) subschema

(iii) virtual table

(iv) none of these

(b) User's view is also called

(i) external view

(ii) conceptual view

(iii) internal view

(iv) none of these

(c) Which of the following schemas defines the stored data structures in terms of the database model used -

- (i) external
- (ii) conceptual
- (iii) internal
- (iv) none of these

(d) Data is processed by using

- (i) DDL
- (ii) DML
- (iii) DCL
- (iv) DPL

(e) Immunity of the conceptual (or external) schemas to changes the internal schemas is referred to as

- (i) physical data independence
- (ii) logical data independence
- (iii) both (i) and (ii)
- (iv) none of these

1.18 Answer to Check Your Progress

1. a. (iv), b. (iv), c. (ii), d. (i), e. (i)
2. (i) F, (ii) T, (iii) F, (iv) F, (v) T, (vi) T
3. a. (ii), b. (i), c. (ii), d. (ii), e. (i)

Unit-2

1.1 Learning Objectives

1.2 Introduction

1.3 What is Data Model

1.4 Need for Data Model

1.5 Types of Data Model

1.6 ER Model

1.7 Check Your Progress

1.8 Answer to Check Your Progress

1.1 Learning Objectives

After going through this unit, you will be able to:

- identify the need for DBMS architecture
- describe the features of DBMS structure
- describe application development approach
- convert E-R diagram to a relational databases and vice versa

1.2 Introduction

A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized and manipulated. The most popular example of a database model is the relational model, which uses a table-based format.

1.3 What is data Model?

In the database design phases, data are represented using a certain data model. The data model is a collection of concepts or notations for describing data, data relationships, data semantics and data constraints. Most data models also include a set of basic operations for manipulating data in the database.

1.4 Need for Data Model?

The purpose of a data model is to represent data and to make the data understandable. There have been many data models proposed in the literature. They fall into three broad categories:

- Object Based Data Models
- Physical Data Models
- Record Based Data Models

The object based and record based data models are used to describe data at the conceptual and external levels, the physical data model is used to describe data at the internal level.

Object Based Data Models

Object based data models use concepts such as entities, attributes, and relationships. An entity is a distinct object (a person, place, concept, and event) in the organization that is to be represented in the database. An attribute is a property that describes some aspect of the object that we wish to record, and a relationship is an association between entities.

Some of the more common types of object based data model are:

- Entity-Relationship
- Object Oriented
- Semantic
- Functional

The Entity-Relationship model has emerged as one of the main techniques for modeling database design and forms the basis for the database design methodology. The object oriented data model extends the definition of an entity to include, not only the attributes that describe the state of the object but also the actions that are associated with the object, that is, its behavior. The object is said to encapsulate both state and behavior. Entities in semantic systems represent the equivalent of a record in a relational system or an object in an OO system but they do not include behaviour (methods). They are abstractions 'used to represent real world (e.g. customer) or conceptual (e.g. bank account) objects. The functional data model is now almost twenty years old. The original idea was to' view the database as a collection of extensionally defined functions and to use a functional language for querying the database.

Physical Data Models

Physical data models describe how data is stored in the computer, representing information such as record structures, record ordering, and access paths. There are not as many physical data models as logical data models, the most common one being the Unifying Model.

Record Based Logical Models

Record based logical models are used in describing data at the logical and view levels. In contrast to object based data models, they are used to specify the overall logical structure of the database and to provide a higher-level description of the implementation. Record based models are so named because the database is structured in fixed format records of several types. Each record type defines a fixed number of fields, or attributes, and each field is usually of a fixed length.

The three most widely accepted record based data models are:

- Hierarchical Model
- Network Model
- Relational Model

The relational model has gained favor over the other two in recent years. The network and hierarchical models are still used in a large number of older databases.

1.5 Types of Data Model

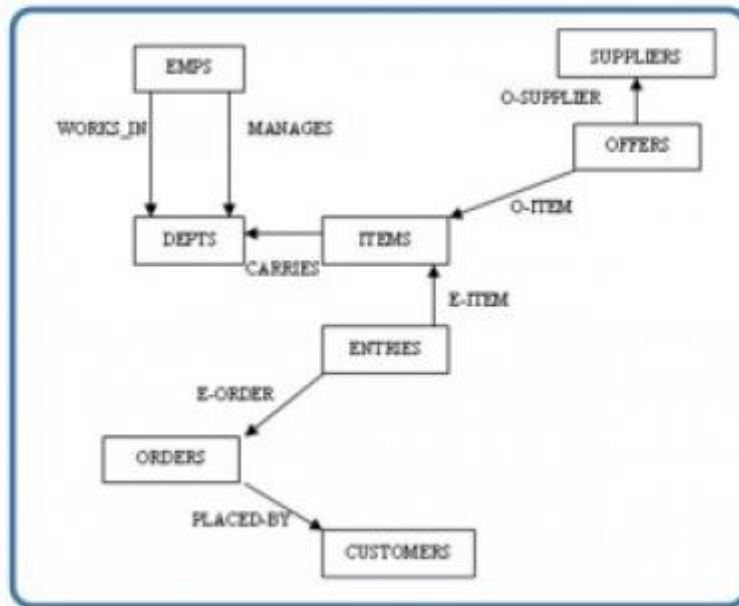
Database systems can be based on different data models or database models respectively. A data model is a collection of concepts and rules for the description of the structure of the database. Structure of the database means the data types, the constraints and the relationships for the description or storage of data respectively.

The most often used data models are

Network Model and Hierarchical Model

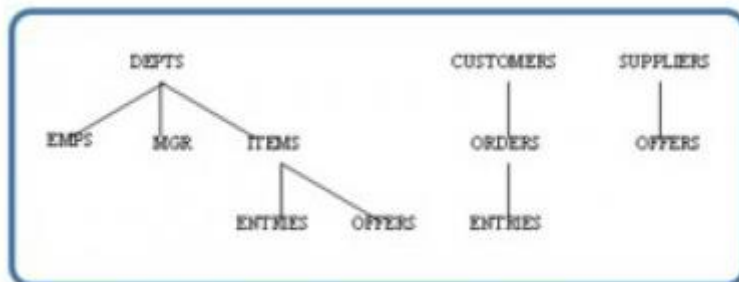
The network model and the hierarchical model are the predecessors of the relational model. They build upon individual data sets and are able to express hierarchical or network like structures of the real world.

- The network model represents data as record types. This model also represents a limited type of one to many relationship called a set type, as shown in Figure.



Network model diagram.

- The hierarchical model represents data as a hierarchical tree structure. Each branch of the hierarchy represents a number of related records. The Following Figure shows this schema in hierarchical model notation.



Hierarchical model diagram.

The relational model

The relational model represents data as relations, or tables. For example, in the membership system at Science World, each membership has many members as shows in the following diagram. The membership identifier, expiry date and address information are fields in the membership. The members are individuals such as Mickey, Minnie, Mighty, Door, Tom, King, Man and Moose. Each record is said to be an instance of the membership table.

Membership

ID EXPIRY DATE Prev Exp Stat Cat

Name Res

Address

City Prov Country

Notes

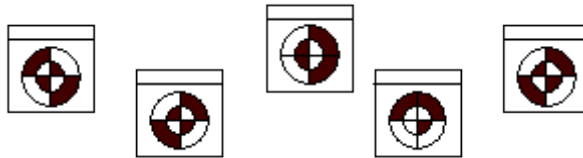
Cards # Members #Years

First Name	Last Name	YYMM	G	BARCODE	V	DATE	TIME	F
Mickey	Mouse	0000	M	10000001	4	20130810	10:12:29	y
Minnie	Mouse	0000	F	10000002	4	20130810	10:12:29	y
Mighty	Mouse	0000	M	10000003	4	20130810	10:12:29	y
Door	Mouse	0000	F	10000004	4	20130810	10:12:29	y
Tom	Mouse	0000	M	10000005	4	20130810	10:12:29	y
King	Rat	0000	M	10000006	4	20130810	10:12:29	y
Man	Mouse	0000	M	10000007	4	20130810	10:12:29	y
Moose	Mouse	0000	M	10000008	4	20130810	10:12:29	y

Record: 1 of 1 | No Filter | Search

Object-oriented Model





Object-oriented models define a database as a collection of objects with features and methods. A detailed discussion of object-oriented databases follows in an advanced module.



Schematic Representation of an Object-oriented Database Model

Object-relational Model

Object-oriented models are very powerful but also quite complex. With the relatively new object-relational database model is the wide spread and simple relational database model extended by some basic object-oriented concepts. These allow us to work with the widely know relational database model but also have some advantages of the object-oriented model without its complexity.

ID	XY	DF	ER
56		XXX	
45		YYY	
...

Schematic Representation of the object-relational Database Model

1.6 The Entity Relationship Data Model

The entity relationship (ER) data model has existed for over 35 years. It is well suited to data modeling for use with databases because it is fairly abstract and is easy to discuss and explain. ER models are readily translated to relations. ER models, also called an ER schema, are represented by ER diagrams.

ER modeling is based on two concepts:

- Entities, defined as tables that hold specific information (data)
- Relationships, defined as the associations or interactions between entities

We will use a sample database called the COMPANY database to illustrate the concepts of the ER model. This database contains information about employees, departments and projects. Important points to note include:

- There are several departments in the company. Each department has a unique identification, a name, location of the office and a particular employee who manages the department.
- A department controls a number of projects, each of which has a unique name, a unique number and a budget.
- Each employee has a name, identification number, address, salary and birthdate. An employee is assigned to one department but can join in several projects. We need to record the start date of the employee in each project. We also need to know the direct supervisor of each employee.
- We want to keep track of the dependents for each employee. Each dependent has a name, birthdate and relationship with the employee.

Entity, Entity Set and Entity Type

An entity is an object in the real world with an independent existence that can be differentiated from other objects. An entity might be

- An object with physical existence (e.g., a lecturer, a student, a car)
- An object with conceptual existence (e.g., a course, a job, a position)

Entities can be classified based on their strength. An entity is considered weak if its tables are existence dependent.

- That is, it cannot exist without a relationship with another entity
- Its primary key is derived from the primary key of the parent entity

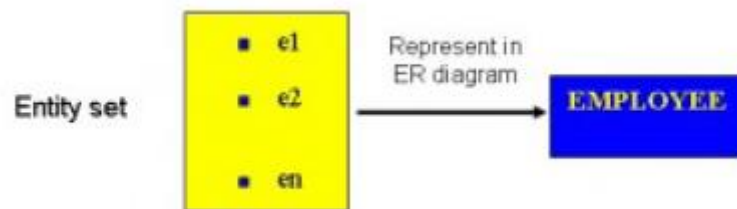
The Spouse table, in the COMPANY database, is a weak entity because its primary key is dependent on the Employee table. Without a corresponding employee record, the spouse record would not exist.

An entity is considered strong if it can exist apart from all of its related entities.

- Kernels are strong entities.
- A table without a foreign key or a table that contains a foreign key that can contain nulls is a strong entity

Another term to know is entity type which defines a collection of similar entities.

An entity set is a collection of entities of an entity type at a particular point of time. In an entity relationship diagram (ERD), an entity type is represented by a name in a box. For example, in the following Figure the entity type is EMPLOYEE.



ERD with entity type EMPLOYEE.

Existence dependency

An entity's existence is dependent on the existence of the related entity. It is existence-dependent if it has a mandatory foreign key (i.e., a foreign key attribute that cannot be null). For example, in the COMPANY database, a Spouse entity is existence -dependent on the Employee entity.

Kinds of Entities

You should also be familiar with different kinds of entities including independent entities, dependent entities and characteristic entities. These are described below.

Independent entities

Independent entities, also referred to as kernels, are the backbone of the database. They are what other tables are based on. Kernels have the following characteristics:

- They are the building blocks of a database.
- The primary key may be simple or composite.
- The primary key is not a foreign key.
- They do not depend on another entity for their existence.

If we refer back to our COMPANY database, examples of an independent entity include the Customer table, Employee table or Product table.

Dependent entities

Dependent entities, also referred to as derived entities, depend on other tables for their meaning. These entities have the following characteristics:

- Dependent entities are used to connect two kernels together.
- They are said to be existence dependent on two or more tables.
- Many to many relationships become associative tables with at least two foreign keys.
- They may contain other attributes.
- The foreign key identifies each associated table.
- There are three options for the primary key:
 1. Use a composite of foreign keys of associated tables if unique
 2. Use a composite of foreign keys and a qualifying column
 3. Create a new simple primary key

Characteristic entities

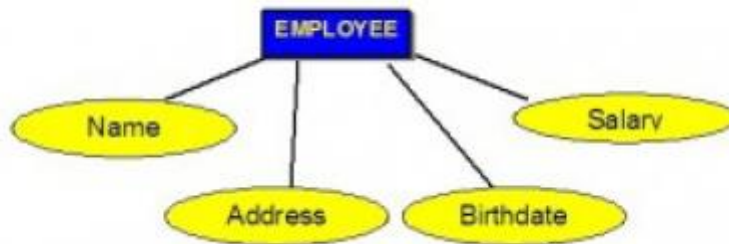
Characteristic entities provide more information about another table. These entities have the following characteristics:

- They represent multivalued attributes.
- They describe other entities.
- They typically have a one to many relationship.
- The foreign key is used to further identify the characterized table.
- Options for primary key are as follows:
 1. Use a composite of foreign key plus a qualifying column
 2. Create a new simple primary key. In the COMPANY database, these might include:
 - Employee (EID, Name, Address, Age, Salary) – EID is the simple primary key.
 - EmployeePhone (EID, Phone) – EID is part of a composite primary key. Here, EID is also a foreign key.

Each entity is described by a set of attributes (e.g., Employee = (Name, Address, Birthdate (Age), Salary).

Each attribute has a name, and is associated with an entity and a domain of legal values. However, the information about attribute domain is not presented on the ERD.

In the entity relationship diagram, shown in the following Figure, each attribute is represented by an oval with a name inside.



How attributes are represented in an ERD.

Types of Attributes

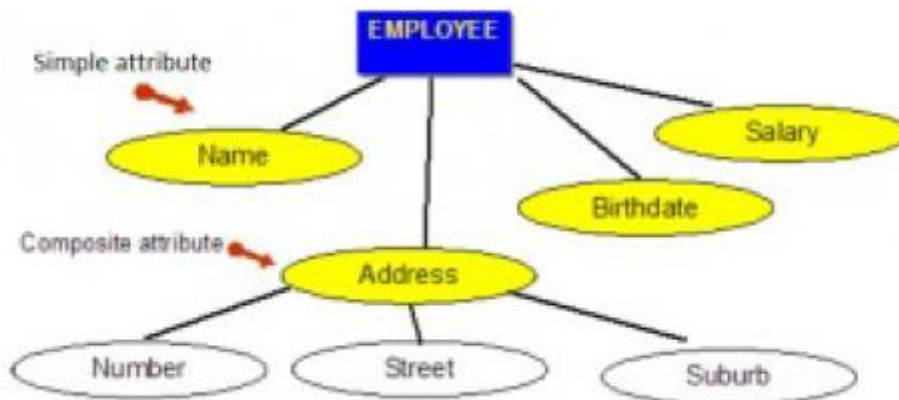
There are a few types of attributes you need to be familiar with. Some of these are to be left as is, but some need to be adjusted to facilitate representation in the relational model. This first section will discuss the types of attributes. Later on we will discuss fixing the attributes to fit correctly into the relational model.

Simple attributes

Simple attributes are those drawn from the atomic value domains; they are also called single-valued attributes. In the COMPANY database, an example of this would be: Name = {John} ; Age = {23}

Composite attributes

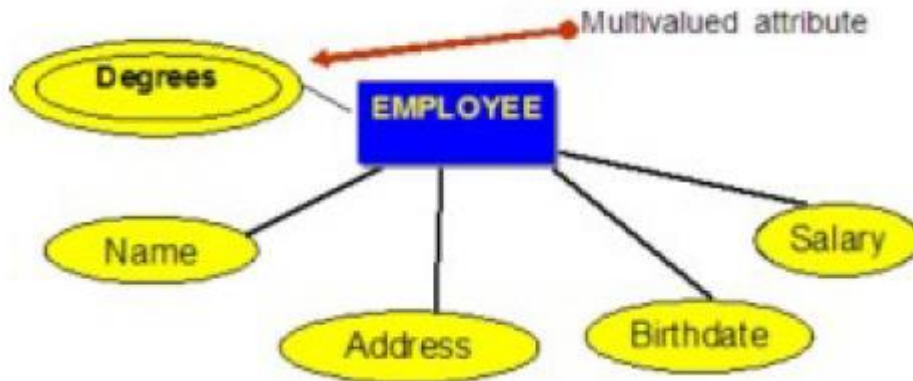
Composite attributes are those that consist of a hierarchy of attributes. Using our database example, and shown in the following Figure, Address may consist of Number, Street and Suburb. So this would be written as → Address = {59 + 'Meek Street' + 'Kingsford'}



An example of composite attributes.

Multivalued attributes

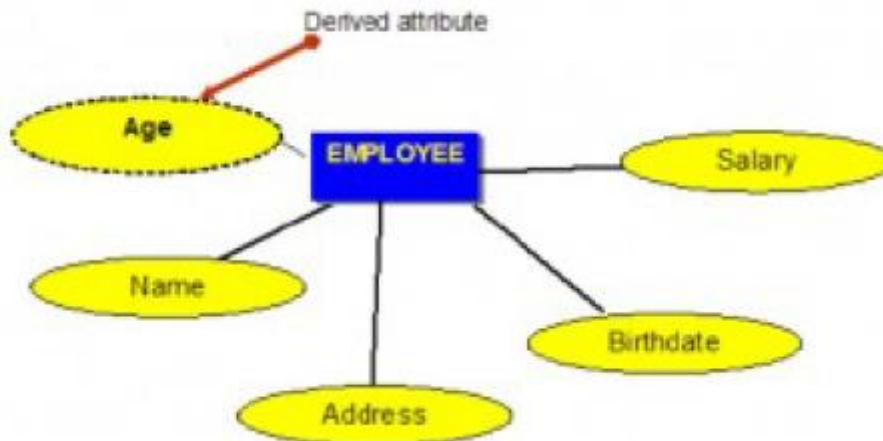
Multivalued attributes are attributes that have a set of values for each entity. An example of a multivalued attribute from the COMPANY database, as seen in the following Figure are the degrees of an employee: BSc, MIT, PhD.



Example of a multivalued attribute.

Derived attributes

Derived attributes are attributes that contain values calculated from other attributes. An example of this can be seen in the following Figure. Age can be derived from the attribute Birthdate. In this situation, Birthdate is called a stored attribute, which is physically saved to the database.



Example of a derived attribute.

Relationships

Relationships are the glue that holds the tables together. They are used to connect related information between tables.

Relationship strength is based on how the primary key of a related entity is defined. A weak, or non-identifying, relationship exists if the primary key of the related entity does not contain a primary key component of the parent entity.

Company database examples include:

- Customer(CustID, CustName)
- Order(OrderID, CustID, Date)

A strong, or identifying, relationship exists when the primary key of the related entity contains the primary key component of the parent entity. Examples include:

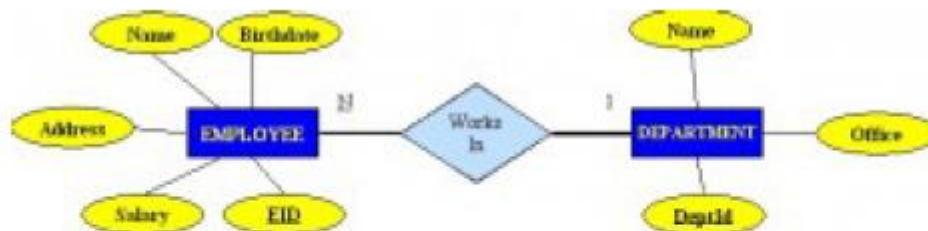
- Course(**CrsCode**, DeptCode, Description)
- Class(**CrsCode**, **Section**, ClassTime...)

Types of Relationships

Below are descriptions of the various types of relationships.

One to many (1:M) relationship

A one to many (1:M) relationship should be the norm in any relational database design and is found in all relational database environments. For example, one department has many employees. In the following Figure shows the relationship of one of these employees to the department.



Relational Schema

EMPLOYEE(EID, Name, Address, Birthdate, Salary, DeptId)

DEPARTMENT(DeptId, Name, Office)

Example of a one to many relationship.

One to one (1:1) relationship

A one to one (1:1) relationship is the relationship of one entity to only one other entity, and vice versa. It should be rare in any relational database design. In fact, it could indicate that two entities actually belong in the same table.

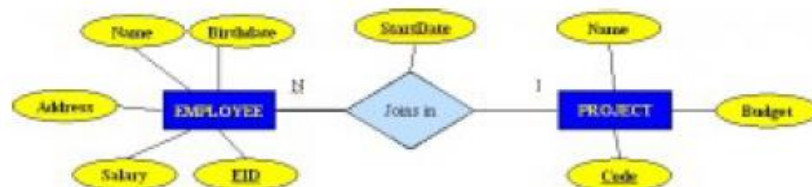
An example from the COMPANY database is one employee is associated with one spouse, and one spouse is associated with one employee.

Many to many (M:N) relationships

For a many to many relationship, consider the following points:

- It cannot be implemented as such in the relational model.
- It can be changed into two 1:M relationships.
- It can be implemented by breaking up to produce a set of 1:M relationships.
- It involves the implementation of a composite entity.
- Creates two or more 1:M relationships.
- The composite entity table must contain at least the primary keys of the original tables.
- The linking table contains multiple occurrences of the foreign key values.
- Additional attributes may be assigned as needed.
- It can avoid problems inherent in an M:N relationship by creating a composite entity or bridge entity. For example, an employee can work on many projects OR a project can have many employees working on it, depending on the business rules. Or, a student can have many classes and a class can hold many students.

The following Figure shows another another aspect of the M:N relationship where an employee has different start dates for different projects. Therefore, we need a JOIN table that contains the EID, Code and StartDate.



Relational Schema
EMPLOYEE (EID, Name, Address, Birthdate, Salary)
PROJECT (Code, Name, Budget)
JOIN(EID, Code, StartDate)

Example where employee has different start dates for different projects.

Example of mapping an M:N binary relationship type

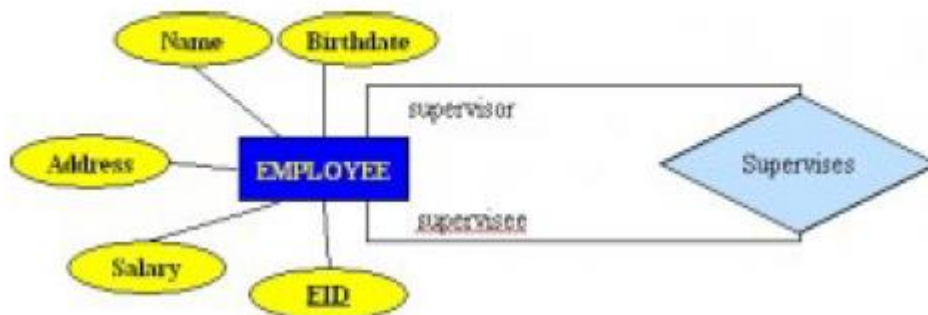
- For each M:N binary relationship, identify two relations.
- A and B represent two entity types participating in R.
- Create a new relation S to represent R.
- S needs to contain the PKs of A and B. These together can be the PK in the S table OR these together with another simple attribute in the new table R can be the PK.
- The combination of the primary keys (A and B) will make the primary key of S.

Unary relationship (recursive)

A unary relationship, also called recursive, is one in which a relationship exists between occurrences of the same entity set.

In this relationship, the primary and foreign keys are the same, but they represent two entities with different roles. See the following Figure for an example.

For some entities in a unary relationship, a separate column can be created that refers to the primary key of the same entity set.



Relational Schema

EMPLOYEE (EID, Name, Address, Birthdate, Salary, Super-EID)

Example of a unary relationship.

Ternary Relationships

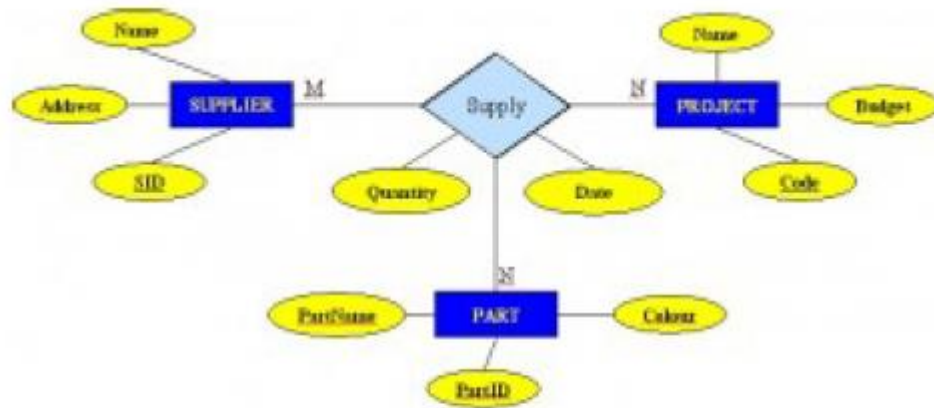
A ternary relationship is a relationship type that involves many to many relationships between three tables.

Refer to the following Figure for an example of mapping a ternary relationship type.

Note n-ary means multiple tables in a relationship. (Remember, N = many.)

- For each **n-ary** (> 2) relationship, create a new relation to represent the relationship.

- The primary key of the new relation is a combination of the primary keys of the participating entities that hold the N (many) side.
- In most cases of an n-ary relationship, all the participating entities hold a **many** side.



Relational Schema

```
SUPPLIER (SID, Name, Address)
PROJECT (Code, Name, Budget)
PART (PartID, PartName, Colour)
Supply (SID, Code, PartID, Quantity, Date)
```

Example of a ternary relationship.

1.7 Check Your Progress

1. State whether the following are true or false (T/F)

- Data model gives logical as well as physical data structure
- Entity relationship model is the example of conceptual data model
- Conceptual data model is also called object-based model.
- High level data model use records as the key data representation component.
- Low level data model describes the details of how to store data in computer.
- Logical data model provide concept which is complicated for end users.
- Implementation data model is also a record-based data model.
- Access path can search database record much faster.
- Network model represents data as hierarchical tree structure.
- Hierarchical model is the current commercial DBMS.

2. State whether the following are true or false (T/F)

- An entity is an object used to represent things.
- Entities having key attributes are called weak entity.
- Strong entity is also called regular entity.
- Domain may be the range of values used against attribute.

e) Composite attribute cannot be divided into smaller parts.

3. Fill in the blanks out of the following: attributes, many-to-many, diamond, association, ER.

- a) An _____ of several entities in an Entity-Relation model is called relationship.
- b) The various kinds of data that describes an entity are known as its _____.
- c) A weak entity set is represented by a doubly outlined rectangle in the _____ diagram.
- d) In an ER diagram a _____ represents a relationship.
- e) A _____ relationship describes entities that may have many relationships in both the directions.

1.8 Answer to Check Your Progress

1.

- i) F ii) T iii) T iv) F v) T
vi) F vii) T viii) T ix) F x) T

2.

- (a) T (b) F (c) T d. T e. F

3.

- (a) association (b) attributes (c) E-R
(d) diamond (e) many-to-many

Unit-3

1.1 Learning Objectives

1.2 Introduction

1.3 What is Relational data model

1.4 Relation, Tuple, Attribute, Cardinality, Degree, Domain

1.5 Check Your Progress

1.6 Answer to Check Your Progress

1.1 Learning Objectives

After going through this unit, the learner will be able to learn:

- About the relational data model
- About Relation, Tuple, Attribute, Cardinality, Degree and Domain

1.2 Introduction

A relational database is a system for storing and accessing data organized into relations. A relation is a bag of tuples. Each tuple is an ordered sequence of attributes. Each attribute is a data value belonging to some data type. All of the tuples in a relation have the same number of attributes. In addition, the relation has a schema that is imposed on each tuple in the relation, specifying what the data type each attribute in each tuple will have. For example, the relation's schema might specify that the first attribute in each tuple is an integer.

1.3 What is Relational data model

The relational data model was introduced by C. F. Codd in 1970. Currently, it is the most widely used data model.

The relational model has provided the basis for:

- Research on the theory of data/relationship/constraint
- Numerous database design methodologies
- The standard database access language called structured query language (SQL)
- Almost all modern commercial database management systems

The relational data model describes the world as “a collection of inter-related relations (or tables).”

Fundamental Concepts in the Relational Data Model

Relation

A relation, also known as a table or file, is a subset of the Cartesian product of a list of domains characterized by a name. And within a table, each row represents a group of related data values. A row, or record, is also known as a tuple.

The columns in a table is a field and is also referred to as an attribute. You can also think of it this way: an attribute is used to define the record and a record contains a set of attributes.

The steps below outline the logic between a relation and its domains.

1. Given n domains are denoted by D_1, D_2, \dots, D_n
2. And r is a relation defined on these domains
3. Then $r \subseteq D_1 \times D_2 \times \dots \times D_n$

Table

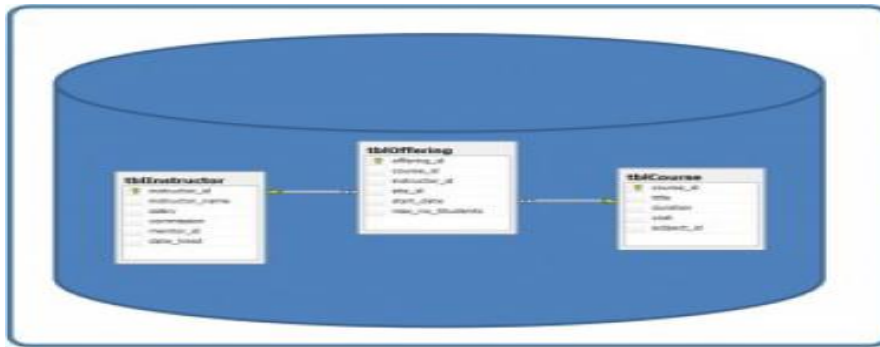
A database is composed of multiple tables and each table holds the data. Figure 7.1 shows a database that contains three tables.

Column

A database stores pieces of information or facts in an organized way. Understanding how to use and get the most out of databases requires us to understand that method of organization.

The principal storage units are called columns or fields or attributes. These house the basic components of data into which your content can be broken down. When deciding which fields to create, you need to think generically about your information, for example, drawing out the common components of the information that you will store in the database and avoiding the specifics that distinguish one item from another.

Look at the example of an ID card in the following Figure to see the relationship between fields and their data.



Database with three tables.

Field Name	Data
First Name	Isabelle
Family Name	Whelan
Nationality	British
Salary	109,900
Date of Birth	15 September 1983
Marital Status	Single
Shift	Mon, Wed
Place of issue	Addis Ababa
Valid until	17 December 2003

Example of an ID card by A. Watt.

Domain

A domain is the original sets of atomic values used to model data. By atomic value, we mean that each value in the domain is indivisible as far as the relational model is concerned. For example:

- The domain of Marital Status has a set of possibilities: Married, Single, Divorced.
- The domain of Shift has the set of all possible days: {Mon, Tue, Wed...}.
- The domain of Salary is the set of all floating-point numbers greater than 0 and less than 200,000.
- The domain of First Name is the set of character strings that represents names of people.

In summary, a domain is a set of acceptable values that a column is allowed to contain. This is based on various properties and the data type for the column.

Records

Just as the content of any one document or item needs to be broken down into its constituent bits of data for storage in the fields, the link between them also needs to be available so that they can be reconstituted into their whole form.

Records allow us to do this. Records contain fields that are related, such as a customer or an employee. As noted earlier, a tuple is another term used for record.

Records and fields form the basis of all databases. A simple table gives us the clearest picture of how records and fields work together in a database storage project.

Record ID	PubDate	Author	Title
1	26/07/1968	B. Pitt	Rights and Wrongs online
2	3/5/2000	A. Jolie	Networking for Change
3	27/02/1971	J. Carter	The Myth of Cyber Crimes
4	15/09/1983	I. Wheaton	Connecting the Disconnected

Example of a simple table by A. Watt.

The simple table example in above Figure shows us how fields can hold a range of different sorts of data. This one has:

- A Record ID field: this is an ordinal number; its data type is an integer.
- A PubDate field: this is displayed as day/month/year; its data type is date.
- An Author field: this is displayed as Initial. Surname; its data type is text.
- A Title field text: free text can be entered here.

You can command the database to sift through its data and organize it in a particular way. For example, you can request that a selection of records be limited by date: 1. all before a given date, 2. all after a given date or 3. all between two given dates. Similarly, you can choose to have records sorted by date. Because the field, or record, containing the data is set up as a Date field, the database reads the information in the Date field not just as numbers separated by slashes, but rather, as dates that must be ordered according to a calendar system.

Degree

The degree is the number of attributes in a table. In our example in above Figure , the degree is 4.

Properties of a Table

- A table has a name that is distinct from all other tables in the database.
- There are no duplicate rows; each row is distinct.
- Entries in columns are atomic. The table does not contain repeating groups or multivalued attributes.
- Entries from columns are from the same domain based on their data type including:
 - number (numeric, integer, float, smallint,...)

- character (string)
 - date
 - logical (true or false)
- Operations combining different data types are disallowed.
 - Each attribute has a distinct name.
 - The sequence of columns is insignificant.
 - The sequence of rows is insignificant.

1.4 Relation, Tuple, Attribute, Cardinality, Degree, Domain

Relation: a subset of the Cartesian product of a list of domains characterized by a name; the technical term for table or file.

A relation is a bag of *tuples*. Each tuple is an ordered sequence of *attributes*. Each attribute is a data value belonging to some data type. All of the tuples in a relation have the same number of attributes. In addition, the relation has a schema that is imposed on each tuple in the relation, specifying what the data type each attribute in each tuple will have. For example, the relation's schema might specify that the first attribute in each tuple is an integer. The relation's schema also gives a name to each attribute. The attribute names give us a convenient way of referring to tuple attributes without having to say "the first attribute", "the fourth attribute", etc

Typical relational databases support numeric and text data types as tuple attributes. Many relational databases also support "BLOBs" (Binary Large Objects) as attributes. A BLOB is a large, uninterpreted chunk of data. BLOBs are useful for storing files, images, and other large chunks of data in a relational database.

Example of relation

Let's say we are going to use a relational database to store information about books. We might define a relation called books to store information about each book:

author_lastname	author_firstname	title	ISBN
Smallfinger	F.G.	A History of Hats	0-651-65165-4
Whittlbey	W.H.J.	Guide to Impossible Buildings	82-234-5475-0
Earwig	Lettice	First Flights in Witchcraft	5-9672-6521-X

author_lastname	author_firstname	title	ISBN
Lightly	W.E.	Habits of the Wolves	91-33-65168-X
Tacticus	Callus	Sieges and Survival	0-651-65165-4
Tacticus	Callus	VENI VIDI VICI: A Soldier's Life	84-15978-99-5

In this relation, there are four attributes called **author_lastname**, **author_firstname**, **title**, and **ISBN**. Each attribute is a text string.

Databases with multiple relations

Databases will typically have many relations. One motivation for allowing multiple relations in a database is to avoid storing redundant information. For example, in the relation above, there are two tuples representing books by the same author, **Callus Tacticus**. Because the author name is represented twice, there is the possibility that this information might not be recorded consistently if the relation were modified.

We can avoid this redundancy by splitting the database into two relations, **books** and **authors**:

The **books** relation:

author_id	title	ISBN
1	A History of Hats	0-651-65165-4
2	Guide to Impossible Buildings	82-234-5475-0
3	First Flights in Witchcraft	5-9672-6521-X
4	Habits of the Wolves	91-33-65168-X
5	Sieges and Survival	0-651-65165-4
5	VENI VIDI VICI: A Soldier's Life	84-15978-99-5

The **authors** relation:

author_id	author_lastname	author_firstname
1	Smallfinger	F.G.
2	Whittlbey	W.H.J.
3	Earwig	Lettice

author_id	author_lastname	author_firstname
4	Lightly	W.E.
5	Tacticus	Callus

The **books** relation has been changed so that the author of each book is represented by a unique integer identifier, the **author_id** attribute. This attribute also exists in the **authors** relation. So, the author of each book tuple in the **books** relation is represented indirectly, by reference to a matching author tuple in the **authors** relation.

Queries, SQL

A *query* is a request to retrieve information from a database.

SQL, the **Structured Query Language**, is a standard language for describing queries in relational databases. SQL is an interesting language because it is *declarative*: it describes *what* information is desired, but does not specify *how* that information is to be retrieved. It is the job of the database to figure out how to find the information requested by a query.

Example: let's say we want to find the titles of all books written by **F.G. Smallfinger**. In our original database, in which only the **books** relation exists, we could express that query as follows:

```
select title
  from books
 where author_lastname = 'Smallfinger' and author_firstname = 'F.G.'
```

A SQL **select** statement specifies a query, and has three parts:

- Which attribute values to retrieve. In the example above, only the **title** attribute is requested.
- Which relations are involved in the query. In the example above, only the **books** relation is queried.
- A *condition* describing which tuples contain the desired information. In the example above, the condition requires that tuples were **author_lastname = 'Smallfinger'** and **author_firstname = 'F.G.'** are desired.

This query will match a single tuple in the **books** relation, and return a single **title** value, **A**

History of Hats.

Joins

A *join* is a query which retrieves information from multiple relations. Joins are a powerful way to exploit *associations* between tuples in different relations. The idea is that a query retrieving information from multiple relations will specify a *join condition* which links attribute values in tuples of two relations.

Let's consider how to find the titles of all books by **F.G. Smallfinger** in the second version of the database, where we have two relations, **books** and **authors**. We will need to do a join of both relations in order to connect the author name and book title, which are now stored in different relations:

```
select books.title
  from books, authors
 where books.author_id = authors.author_id
        and authors.author_lastname = 'Smallfinger' and
authors.author_firstname = 'F.G.'
```

Note several interesting details of this query:

- We are selecting **books.title** as the attribute to retrieve, explicitly noting that this attribute exists in the **books** relation
- The **from** clause now specifies two relations, **books** and **authors**
- The first part of the **where** clause, **books.author_id = authors.author_id**, is the join condition. It states that when considering pairs of tuples in the **books** and **authors** relations as candidates for retrieval, each tuple must contain the same value for the respective **author_id** attributes.
- In the **where** clause, each attribute is qualified by the name of the relation it is a part of. This is especially important when two relations have identically-named attributes, as is the case with the **author_id** attributes of the **books** and **authors** relations.

Indices

Obviously, a database system must be able to answer all queries by returning the requested information.

An important additional characteristic that database systems should have is that queries are answered *efficiently*, even if the database contains a large amount of data.

An *index* is an auxiliary data structure associated with a relation to increase the efficiency of queries on that relation. Specifically, an index is an auxiliary data structure which, given an

attribute value or range of attribute values, quickly locates the tuple or tuples which match that value or range. An index may be applied to a single attribute, or to multiple attributes.

The idea is that an index helps the database focus the search for data matching a query by narrowing the number of tuples that must be checked.

Sequential scans

Consider queries involving a single relation. Any such query can be answered by performing a *sequential scan* over the tuples in the relation: checking each tuple to see if it matches the condition(s) specified in the **where** clause, and if so, returning the values of the selected attributes.

For very large relations, a sequential scan may do much more work than necessary. In particular, the **where** clause may have sub-conditions that are met by only a very small number of tuples: we say that a condition matched by only a small number of tuples has high *selectivity*.

To answer queries as efficiently as possible, the database should limit its scan to as small a subset of tuples as possible. This can be done using an *index*.

For example: let's say that in the book database, we will frequently need to search for authors by last name. To make those queries more efficient, we can build an index on the **author_lastname** attribute. Queries such as

```
select author_id from authors where author_lastname = 'Smith';
```

can be answered efficiently because the index on **author_lastname** will allow the database to ignore tuples in which the value of that attribute is not 'Smith'.

Tuple

Tuple is a technical term for row or record. A relation is defined as a set of tuples that have the same attributes. A tuple usually represents an object and information about that object. Objects are typically physical objects or concepts. A relation is usually described as a table, which is organized into rows and columns. All the data referenced by an attribute are in the same domain and conform to the same constraints.

A relation is defined as a set of tuples that have the same attributes. A tuple usually represents an object and information about that object. Objects are typically physical objects or concepts. A relation is usually described as a table, which is organized into rows and columns. All the data referenced by an attribute are in the same domain and conform to the same constraints.

The relational model specifies that the tuples of a relation have no specific order and that the tuples, in turn, impose no order on the attributes. Applications access data by specifying

queries, which use operations such as select to identify tuples, project to identify attributes, and join to combine relations. Relations can be modified using the insert, delete, and update operators. New tuples can supply explicit values or be derived from a query. Similarly, queries identify tuples for updating or deleting.

Tuples by definition are unique. If the tuple contains a candidate or primary key then obviously it is unique; however, a primary key need not be defined for a row or record to be a tuple. The definition of a tuple requires that it be unique, but does not require a primary key to be defined. Because a tuple is unique, its attributes by definition constitute a superkey.

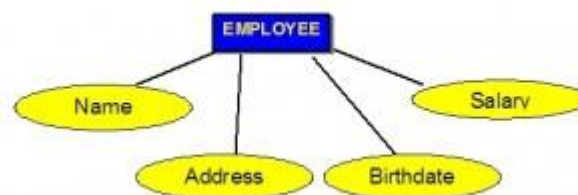
Attribute

An **attribute** is a piece of information about the entity. For example, the title, author, and ISBN are all attributes of a book entity.

Each entity is described by a set of attributes. E.g. Employee = (Name, Address, Age, Salary).

Each attribute has a name, associated with an entity and is associated with a domain of legal values. However the information about attribute domain is not presented on the ER diagram.

In the diagram, each attribute is represented by an oval with a name inside.



Cardinality

Cardinality describes the relationship between two data tables by expressing the minimum and maximum number of entity occurrences associated with one occurrence of a related entity. In the following Figure, you can see that cardinality is represented by the innermost markings on the relationship symbol. In this figure, the cardinality is 0 (zero) on the right and 1 (one) on the left.

The outermost symbol of the relationship symbol, on the other hand, represents the connectivity between the two tables. Connectivity is the relationship between two tables, e.g., one to one or one to many. The only time it is zero is when the FK can be null. When it comes to participation, there are three options to the relationship between these entities:

either 0 (zero), 1 (one) or many. In the following Figure, for example, the connectivity is 1 (one) on the outer, left-hand side of this line and many on the outer, right-hand side.

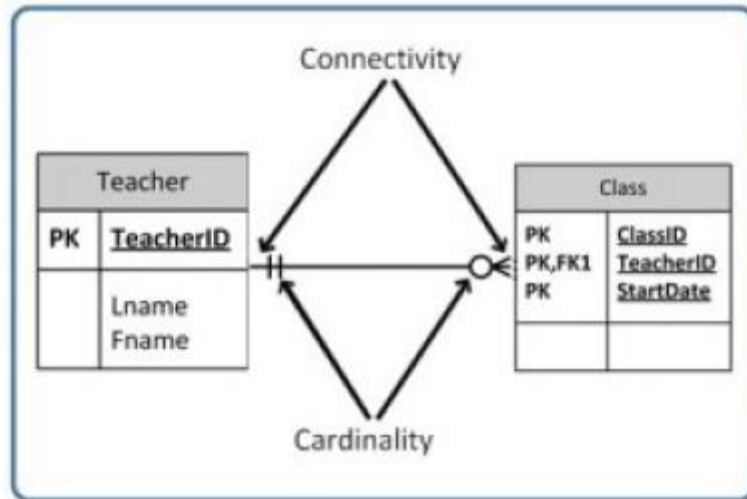


Figure: Position of connectivity and cardinality on a relationship symbol, by A. Watt.

The following Figure shows the symbol that represents a one to many relationship.



In the following Figure, both inner (representing cardinality) and outer (representing connectivity) markers are shown. The left side of this symbol is read as minimum 1 and maximum 1. On the right side, it is read as: minimum 1 and maximum many.



Degree

Degrees are defined prior to people being awarded them. The degree of relationship is the number of occurrences in one entity which are associated to the number of occurrences in another. There are three degrees of relationship, known as:

1. one-to-one
2. one-to-many
3. many-to-many

Domain

In data management and database analysis, a data domain refers to all the values which a data element may contain. The rule for determining the domain boundary may be as simple as a data type with an enumerated list of values.

For example, a database table that has information about people, with one record per person, might have a "gender" column. This gender column might be declared as a string data type,

and allowed to have one of two known code values: "M" for male, "F" for female—and NULL for records where gender is unknown or not applicable (or arguably "U" for unknown as a sentinel value). The data domain for the gender column is: "M", "F".

1.5 Check Your Progress

1. fill in the blanks
 - a. Ais a system for storing and accessing data organized into relations.
 - b. Anis a piece of information about the entity
 - c. A..... is a query which retrieves information from multiple relations.
 - d. Ais a request to retrieve information from a database.
 - e. A database is composed of multipleand each table holds the data

1.6 Answer to Check Your Progress

- a. relational database
- b. attribute
- c. *join*
- d. *query*
- e. tables

Block-2

Unit-1

- 1.1 Learning Objectives
- 1.2 Introduction
- 1.3 Characteristic of SQL
- 1.4 Basic Structure of SQL Queries
- 1.5 Basic Data Types
- 1.6 SQL Commands
- 1.7 Useful Relational Operator
- 1.8 Aggregate Functions
- 1.9 SUM function
- 1.10 AVG Function
- 1.11 Check Your Progress
- 1.12 Answer to Check Your Progress

1.1 Learning Objectives

After going through this unit, the learner will be able to learn:

- To define the SQL data definition
- Basic Structure of SQL Queries
- Set Operation
- Aggregate Functions
- SQL COMMANDS

1.2 Introduction

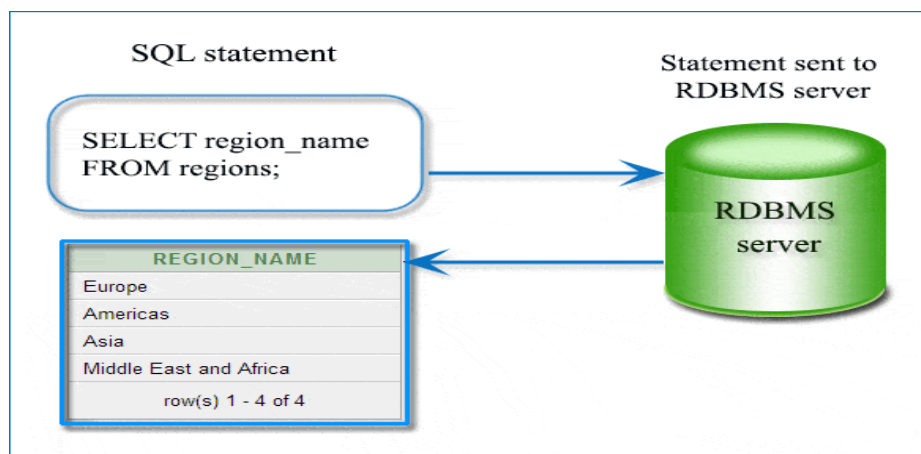
In June 1970 Dr. E. F. Codd published the paper, "A Relational Model of Data for Large Shared Data Banks" in the Association of Computer Machinery (ACM) journal. Codd's model is now accepted as the definitive model for relational database management systems (RDBMS).

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. We human beings communicate with each other with the help of language. Similarly, *SQL* stands for *Structured Query Language* is the language that a database understands, and we will communicate to the database using SQL. SQL is a 4TH generation database gateway language standardized by ANSI (American National Standards Institute) for managing data held in a **RDBMS** (Relational Database Management Systems).

What is SQL?

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.



1.3 Characteristics of SQL

The following are the important features of SQL:

- I. SQL can be used by a range of users, including those with little or no programming experience.
- II. It is non procedural language.
- III. It reduces the amount of time required for creating and maintaining systems.
- IV. It is an English-like language.

1.4 Basic Structure of SQL Queries

The database in a relational database management system consists of a collection of database tables. The basic structure of an SQL query contains of three clause, **select**, **from**, and **where**.

The **select** clause lists the desired attributes of the relation(s) in the result of the query.

The **from** clause lists the relation(s) to be scanned in the evaluation of the expression.

The **where** clause consists of a predicate involving attributes of the relations that appear in the from clause.

A typical SQL query has the following form :

```
select A1,A2,.....,An  
from r1, r2,.....,rm  
where P
```

Each A_i represents an attribute and each r_i represents a relation. P is a predicate.

1.5 Basic Data Types

Each column in an RDBMS table specifies/declares the types of data that the columns stores. This enables RDBMS to use storage space more efficiently by internally storing different types of data in different ways. ANSI SQL includes the following basic **data types**.

1. CHARACTER

- **CHARACTER(n)** or **CHAR(n)**: Fixed width n-character string, padded with spaces as needed.
- **CHARACTER VARYING(n)** or **VARCHAR(n)**: Variable width string with a maximum size of n characters.
- **NATIONAL CHARACTER(n)** or **NCHAR(n)**: Fixed width n-character string supporting an international character set(Unicode Character).
- **NATIONAL CHARACTER VARYING(n)** or **NVARCHAR(n)**: Variable width string with a maximum size of n characters supporting an international character set.

2. NUMERIC

- **SMALLINT, INTEGER OR INT**

- **FLOAT, REAL and DOUBLE PRECISION**

- **NUMERIC**(precision, scale) or **DECIMAL**(precision, scale) (e.g. 1234.56)

The number 1234.56 has a precision of 6 and a scale of 2. A scale of 0 indicates that the number is an integer.

3. DATE

- **DATE**: Date values (e.g. 1999-12-31).

- **TIME**: Time values (e.g. 23:30:10). The granularity of the time value is usually a tick (100 nanoseconds).

- **TIMESTAMP**: Date and a Time put together (e.g. 1999-12-31 23:30:10).

4. BIT

- **BIT**: bit values (e.g. 0 or 1)

Additional NON-ANSI **LARGE OBJECT** data types - CLOB, BLOB, LONG, RAW, LONG RAW (Oracle specific); VARBINARY(n) (MSSQL Server specific)

1.6 SQL Commands

The scope of SQL includes schema creation and modification, data access control, data insert, query, update and delete. SQL Commands are broadly classified into the following categories:

Data Definition Language (DDL): DDL stands for Data Definition Language. These statements are used to define the database structure (also known as database schema). Given below are the DDL statements:

- **CREATE** – CREATE is used for creating database objects, such as tables, views, indexes, etc.
- **DROP** – DROP is used for deleting database objects.
- **ALTER** – ALTER is used for modifying the structure of database objects.
- **RENAME** – RENAME is used for renaming database objects.
- **TRUNCATE** – TRUNCATE is used for deleting all records from a table.
- **COMMENT** – COMMENT is used for adding comments to a data dictionary.

Data Manipulation Language (DML): These statements are used to manage data within database objects. Given below are the DML commands:

- SELECT – SELECT is used to retrieve data from a database.
- INSERT – INSERT is used to insert data into a table.
- UPDATE – UPDATE is used to update existing data within a table.
- DELETE – DELETE is used to delete all records from a table, the space for the records remain.
- MERGE – MERGE is used to UPSERT operation (conditional INSERT/UPDATE).
- CALL – CALL is used to call a PL/SQL or Java subprogram.
- EXPLAIN PLAN – EXPLAIN PLAN is used to explain access path to data.
- LOCK TABLE – LOCK TABLE is used to control concurrency.

Data Query Language (DQL): Data Query Language (DQL) mainly deals with SQL SELECT statement for retrieving data from a database

- SELECT – SELECT command is used to select statement for retrieving data from a database

Transaction Control Language (TCL): The TCL statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.

- COMMIT – COMMIT saves the work done.
- SAVEPOINT – SAVEPOINT identifies a point in a transaction to which you can later roll back.
- ROLLBACK – ROLLBACK restores database to original state since the last COMMIT.
- SET TRANSACTION – SET TRANSACTION changes transaction options like isolation level and what rollback segment to use.

Data Control Language (DCL): DCL stands for Data Control Language. Given below are the DCL statements:

- GRANT – GRANT command gives user's access privileges to database.
- REVOKE – REVOKE command withdraws access privileges given with the GRANT command.

1.7 USEFUL RELATIONAL OPERATORS

In a Relational Database environment, operators are needed to derive information from the data stored in the database tables. As the tables are set of rows and columns, the relational operators should operate on sets. That is why some of the classical set theory operators like UNION, INTERSECTION, EXCEPT and JOIN operators also show up as relational operators.

UNION operator : The UNION relational operator allows to join information from two or more tables that have the same structure. Tables with same structure means :

The tables must have the same number of columns.

The corresponding columns must have identical data types and lengths.

The syntax for the SQL UNION is as :

```
SELECT column_name(s) FROM table_name1
UNION
SELECT column_name(s) FROM table_name2
```

The union of two tables returns all the rows that appear in either table. The duplicate rows are eliminated. To allow the duplicate values the UNION ALL operator is used. The syntax of which is as :

```
SELECT column_name(s) FROM table_name1
UNION ALL
SELECT column_name(s) FROM table_name2
```

Suppose there are two tables called EMP and CUST having three columns each and also suppose the data types of each of the corresponding columns in the tables are the same. Suppose the table have the following data in them.

SELECT * FROM EMP;

First Name	Last Name	Age
Manab	Nath	32
Rahul	Sarma	40
Binod	Moshahary	35

SELECT * FROM CUST;

First Name	Last Name	Age
Hemant	Rajkonwar	36
Manab	Nath	32
Manash	Kalita	25

Now, the Union of the above two tables displays a virtual result table containing all rows of the first table as well as all the rows of the second table.

```
SELECT * FROM EMP
UNION
SELECT * FROM CUST;
```

The result table will be as follows:

First Name	Last Name	Age
Binod	Moshahary	35
Hemant	Rajkonwar	36
Manab	Nath	32
Manash	Kalita	25
Rahul	Sarma	40

INTERSECTION operator : The SQL INTERSECT operator also operates on two tables but unlike the UNION operator, it returns only those rows that appear in both the tables. It removes duplicate rows from the final result table. The INTERSECT ALL operator does not remove duplicate rows from the result table. The syntax for the INTERSECT operator is as :

```
[SQL statement1]
INTERSECT
[SQL statement2]
```

For example,

```
SELECT * FROM EMP
INTERSECT
SELECT * FROM CUST;
```

The result table would look as follows:

First Name	Last Name	Age
Manab	Nath	32

EXCEPT operator : The SQL EXCEPT operator returns all rows from a database table that appears in the first table but that do not appear in the second table.

Example :

```
SELECT * FROM EMP
EXCEPT
SELECT * FROM CUST;
```

The output table would be as follows:

First Name	Last Name	Age
Binod	Moshahary	35
Rahul	Sarma	40

JOIN operator: The JOIN operator is a powerful relational operator which can combine data from multiple tables. Tables are joined on the columns that have the same data type and width. SQL supports different types of JOIN operations — INNER, OUTER and CROSS.

The **INNER JOIN** is also known as Equi Join. This is because the SELECT WHERE statement generally compares two columns of two different tables with the equivalence operator '='. The syntax for this type of Join is as follows:

```
SELECT column_name(s)
FROM table_name1
INNER JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

Suppose the PRODUCT table has the following data:

Prod_id	prod_name	prod_qty
P1	Shirt	100
P2	Trouser	100
P3	T-shirt	80

SALES table:

cust_id	Prod_id	quantity
C1	P1	35
C5	P3	55

Now, to list the quantity of the products sold, the SQL statement would be as :

```
SELECT Product.prod_name, Sales.quantity
FROM Product
INNER JOIN Sales
ON Product.Prod_id=Sales.Prod_id
```

The result table would be as

Prod_name	quantity
-----------	----------

Shirt	35
T-shirt	55

OUTER JOIN is similar to INNER JOIN but is more flexible in selecting the data from the tables. This type of Join is used to select data or rows from the table on the left or right or both regardless of whether the other table has common values.

The syntax for LEFT Join is :

```
SELECT column_name(s)
FROM table_name1
LEFT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

Suppose to list all the products along with the number of quantities sold, the SQL statement would be as :

```
SELECT Product.Prod_name, Product.prod_qty,
Sales.quantity
FROM Product
LEFT JOIN Sales
ON Product.Prod_id=Sales.Prod_id
```

The result table would be as :

prod_name	prod_qty	quantity
Shirt	100	35
Trouser	100	
T-shirt	80	55

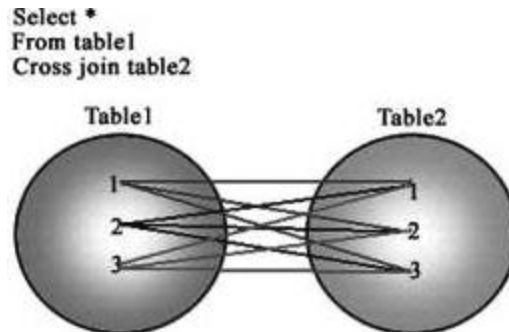
The LEFT Join returns all the rows from the left table even if there are no matches in the right table.

The RIGHT Join returns all the rows from the right table even if there are no matches in the left table. The syntax is as follows:

```
SELECT column_name(s)
FROM table_name1
RIGHT JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

CROSS JOIN returns a Cartesian product. Cartesian product means it returns the number of rows that is equal to the product of all rows in both the tables being joined. That is, it

combines every row from the left table with every row from the right table. For example, if the first table has 20 rows and the second table has 10 rows, the result will be $20 * 10$, or 200 rows. This type of query takes a long time to execute. The pictorial representation of cross join syntax is as:



1.8 AGGREGATE FUNCTIONS

The database tables created with SQL commands stores large amount of data. There are some aggregate functions in SQL to assist in the summarization of the large volumes of data. These aggregate functions in SQL returns a single value, calculated from the values in the column.

Useful aggregate functions are as follows:

AVG() ——— Returns the average value

COUNT() -- Returns the number of rows

FIRST() ---- Returns the first value

LAST() ----- Returns the last value

MAX() ----- Returns the largest value

MIN() ----- Returns the smallest value

SUM() ----- Returns the sum

1.9 SUM Function

The **SUM()** function returns the total sum of the numeric values of a column. The syntax is as follows:

SELECT SUM <column_name> **FROM** <tablename>;

To understand this function better, let us take an example. Suppose we have the following data in the *Products* table.

PROD_ID	PROD_NAME	PROD_QTY
P1	T-Shirts	200
P2	Jeans	150
P3	Trousers	100

P4	Pull Overs	80
P5	Shirts	200

SELECT SUM (prod_qty) “Total Quantity” **FROM** Product;

The result will be the total number of product quantities in the Product table. So, the output of the above statement will be

Total Quantity
730

1.10 AVG Function

The **AVG()** function calculates the average of the values in a specified column. The syntax is as follows:

SELECT AVG <column_name> **FROM** <tablename>;

Let us take an example. Suppose the Employee table contains the following data.

Emp_id	Emp_name	Phone	Sex	Salary
E1	Rani Patil	9856723451	F	34000
E2	Nirmali Das	9546751289	F	20000
E3	Binod Das	8856359341	M	45000
E4	Manav	9506781256	M	27000

SELECT AVG (salary) “Average” **FROM** Employee;

The output of the above statement will be as follows:

Average
31500

1.11 Answer to Check Your Progress

1. Fill in the blanks

- a. Thefunction returns the total sum of the numeric values of a column.
- b. The database tables created with SQL commands stores large amount of.....
- c. **OUTER JOIN** is similar tobut is more flexible in selecting the data from the tables.
- d. Theoperator is a powerful relational operator which can combine data from multiple tables.
- e.are set of rows and columns

- f. The database in adatabase management system consists of a collection of database tables.

1.12 Answer to Check Your Progress

- a. SUM()
- b. Data
- c. INNER JOIN
- d. JOIN
- e. Tables
- f. relational

Unit -2

- 1.1 Learning Objectives
- 1.2 Introduction
- 1.3 Compound Conditions and Logical Operators
- 1.4 AND Operator
- 1.5 OR Operator
- 1.6 Combining AND and OR Operators
- 1.7 IN Operator
- 1.8 BETWEEN Operator
- 1.9 NOT Operator
- 1.10 Order of Precedence for Logical Operators
- 1.11 LIKE Operator
- 1.12 Concatenation Operator
- 1.13 Alias Column Names
- 1.14 ORDER BY Clause
- 1.15 Handling NULL Values
- 1.16 Check Your Progress
- 1.17 Answer to Check Your Progress

1.1 Learning Objectives

After going through this unit, you will be able to:

- learn how to manipulate the data in the database tables
- describe different types of operators used to handle the data
- illustrate the meaning of NULL values and also about how to handle them

1.2 Introduction

We have already learnt in the previous unit about the basics of Structured Query Language (SQL). We have already learnt the creation, insertion and deletion of tables and data of the tables. SQL can also execute queries against a database and can retrieve data from the databases. Updating of data in tables can also be performed in the tables.

In this unit we will discuss the operators that are used to operate the data in the database tables. Operators are used in almost every SQL statement. They are used to compare,

evaluate or calculate values. They tell SQL how to evaluate an SQL expression or a conditional statement. We will find here that operators are mostly used in the WHERE clause of the SQL statement.

1.3 Compound Conditions and Logical Operators

Compound conditions are simple conditions that are joined by logical operators. An *operator* is a symbol specifying an action that is performed on one or more expressions. Logical operators test for the truth of some condition. Most operators will appear inside conditional statements in the **WHERE** clause of SQL Commands. The logical operators in SQL are as follows:

AND, OR and NOT.

The syntax for a compound condition is as follows:

```
SELECT <column_name >
FROM <table_name >
WHERE <simple condition >
{[AND|OR] simple condition};
```

1.4 AND Operator

SQL **AND** joins together two or more conditions and returns result only when all of the conditions are true. **AND** helps to query for very specific records. There is no limit to the number of **AND** conditions that can be applied to a query by utilizing the **WHERE** clause.

For example, the following query will find out only the rows where the customer identity is 'c10' and the product identity is 'p5'. It will not find a row for customer identity 'c12' and product identity p5.

```
SELECT *
FROM Sales
WHERE cust_id='c10'
AND prod_id='p5';
```

The output of the above query will be as follows:

Cust_id	Prod_id	Quantity
c10	p5	25

This example illustrates how SQL **AND** combines multiple conditional statements into a single condition.

1.5 OR Operator

Logical OR compares two Booleans as expression and returns TRUE when either of the conditions is TRUE and returns FALSE when both are FALSE. otherwise, returns UNKNOWN (an operator that has one or two NULL expressions returns UNKNOWN).

Example :

To get data of 'cust_code', 'cust_name', 'cust_city', 'cust_country' and 'grade' from the 'customer' with following conditions -

1. either 'cust_country' is 'USA',
2. or 'grade' of the 'customer' is 3,

the following SQL statement can be used :

```
SELECT cust_code,cust_name, cust_city,cust_country,grade
FROM customer
WHERE cust_country = 'USA' OR grade = 3;
```

Output :

CUST_CODE	CUST_NAME	CUST_CITY	CUST_COUNTRY	GRADE
C00001	Micheal	New York	USA	2
C00020	Albert	New York	USA	3
C00002	Bolt	New York	USA	3
C00010	Charles	Hampshair	UK	3
C00012	Steven	San Jose	USA	1
C00009	Ramesh	Mumbai	India	3
C00011	Sundariya	Chennai	India	3

1.6 Combining AND and OR Operators

The statements in SQL can be combined using the logical operators AND and OR. SQL engine will display the results when **all** the conditions specified with the AND operator

are satisfied and when **any** of the conditions specified with the OR operator are specified. Let us take an example to understand how to combine these operators together into a single statement.

Suppose the '*Employee*' table contains the following data in it:

emp_id	emp_name	position	phone	sex	salary
e1	John	Manager	9876412345	M	55000
e2	Rahul	Asst. Manager	8812534675	M	34000
e3	Jutika	Manager	9853510293	F	55000
e4	Kamal	AGM	8812356473	M	80000
e4	Sona	Executive	6723812345	F	20000

Fig : Table 1

Now, combining the AND and OR operators into a single statement,

```
SELECT emp_id, emp_name
FROM Employee
WHERE Position = 'Manager' AND sex='M'
OR Salary < 80000
```

This statement would return all those records where the employee's position is manager and is a male employee. It will also return those records where the salary of the employee is less than 80000.

1.7 IN Operator

The IN operator is a comparison operator. It is used to compare a value to a list of values that has been specified. It returns TRUE if the compared value matches at least one of the values in the list. The syntax of the IN operator in SQL is as follows:

```
SELECT <column_name(s)>
FROM < table_name >
WHERE <column_name> IN (value1,value2,...);
```

Suppose we have the following data in the *PRODUCTS* table.

PROD ID	PROD_NAME	PROD_QTY
P1	T-Shirts	200
P2	Jeans	150
P3	Trousers	100
P4	Pull Overs	80

P5	Shirts	200
----	--------	-----

Suppose from the above table, we need the records of the product Id's P2, P4, P5. The SQL statement for this query is as follows:

```

SELECT *
FROM Products
WHERE prod_id IN ('P2','P4','P5');

```

The result-set will look like as follows:

PROD_ID	PROD_NAME	PROD_QTY
P2	Jeans	150
P4	Pull Overs	80
P5	Shirts	200

1.8 BETWEEN Operator

The BETWEEN operator is used with a WHERE clause to test whether a value lies in a specified range of values. x is BETWEEN a and b means that $x \geq a$ and $x \leq b$. This is also a comparison operator. The syntax of BETWEEN operator is as follows:

```

SELECT <column_name(s)>
FROM <table_name>
WHERE <column_name>
BETWEEN value1 AND value2

```

Suppose the 'EMPLOYEE' table contains the following data in it:

Emp_id	emp_name	Phone	Sex	Salary
E1	Kabita	9876412345	F	20000
E2	Babita	8812534675	F	15000
E3	Neel	9853510293	M	25000
E4	Konika	8812356473	F	16000
E5	Manoj	6723812345	M	17000

The following example shows all the Employee details that gets salary between Rs 16000 and Rs 25000.

```
SELECT * FROM Employee  
WHERE Salary BETWEEN 16000 and 20000;
```

The result of the above query will be as given below:

Emp_id	emp_name	Phone	Sex	Salary
E1	Kabita	9876412345	F	20000
E3	Neel	9853510293	M	25000
E4	Konika	8812356473	F	16000
E5	Manoj	6723812345	M	17000

1.9 NOT Operator

The NOT operator is used to display only those records which do not satisfy the given conditions. Suppose from the EMPLOYEE table we want to list the details of the employees who are not male. The SQL statement for this query will be as follows:

```
SELECT *  
FROM EMPLOYEE  
WHERE NOT sex='M'
```

The result of the above statement will be as follows:

Emp_id	emp_name	Phone	Sex	Salary
E1	Kabita	9876412345	F	20000
E2	Babita	8812534675	F	15000
E4	Konika	8812356473	F	16000

The NOT operator can also be used with a comparison operator to negate the result of the comparison.

```
NOT BETWEEN  
NOT IN (value1, value2, value3,...)  
NOT LIKE
```

1.10 ORDER OF PRECEDENCE FOR LOGICAL OPERATORS

Operator precedence determines the sequence in which the operations are performed in an SQL expression with multiple operators. While evaluating the SQL statements with multiple operators, the operators with higher precedence are evaluated first before evaluating those with lower precedence.

When more than one logical operators are present in an SQL expression, NOT is evaluated first, then AND and finally OR.

Order of Precedence for logical operators is shown in the table below. Arithmetic operators and comparison operators take higher precedence than logical operators.

Order	Evaluated	Operator
1	Arithmetic operators	
2	Comparison operators	
3	NOT	
4	AND	
5	OR	

1.11 LIKE Operator

The LIKE operator is used with a WHERE clause to compare a character string to a specified pattern. It also is a comparison operator. It searches for the specified pattern in a column of a database table. This is achieved by using wildcard characters. The two wildcard characters are —

% – allows to match strings of any length, that is zero or more characters.

_ – allows to match a single character.

The syntax for this operator is as follows:

```
SELECT <column names>
FROM <tablename>
WHERE <column_name> LIKE <pattern>;
```

Example 1 : List the employees whose names start with the letter 'K'.

The SQL statement would be as follows:

```
SELECT emp_id,emp_name FROM Employee
WHERE emp_name LIKE 'K%';
```

1.12 ALIAS COLUMN NAMES

When data from a database is selected, the heading of the column is same in the output as the column name in the database. The column heading can be changed in the output, that is, a different column heading can be given to a column in a table with the help of the alias column name of SQL. The alias column name can be assigned in the column list of the Select statement using the AS operator.

The syntax for alias column names is as given below:

```
SELECT column_name AS alias_name  
FROM table_name
```

For Example: To display the names of the employees from the Employee table using alias column name, the following SQL statement is written.

```
SELECT emp_name AS Name  
FROM Employee;
```

The output of the above statement would be as follows:

```
Name  
Kabita  
Babita  
Neel  
Konika  
Manoj
```

1.13 ORDER BY Clause

The ORDER BY clause is used to display the output table of a query in either ascending or descending alphabetical order. It sorts the individual rows of a table. *The syntax for the ORDER BY clause is:*

```
SELECT <column_names>  
FROM <Tablename>  
WHERE <predicates>  
ORDER BY <column_name>;
```

The default sort order for ORDER BY clause is an ascending list, [a-z] for characters and [0-9] for numbers. SQL can also sort the records in descending order, [z-a]. The syntax is :

```
SELECT <column_names>  
FROM <Tablename>  
WHERE <predicates>
```


ORDER BY <column_name> DESC;

Example 1:

```
SELECT emp_name, salary
FROM Employee
ORDER BY salary ASC;
```

This would return the records sorted by the salary field in ascending order as given below:.

emp_name	salary
Babita	15000
Konika	16000
Manoj	17000
Kabita	20000
Neel	25000

Example 2:

```
SELECT emp_name, salary
FROM Employee
ORDER BY salary DESC;
```

This would return all the records sorted by the salary field in descending order as given below:

emp_name	salary
Neel	25000
Kabita	20000
Manoj	17000
Konika	16000
Babita	15000

1.14 Handling NULL Values

NULL value means unknown or missing data value. SQL treats any zero-length string like a NULL value. Sometimes there may be records in a table containing no value. This may be because during the data entry time the data was not available or for some rows in a table that particular field is not applicable. A table column, by default, can have NULL values. The NULL value is different from other values like a blank or a zero as zero is a numeric value and a blank space is a character value.

NULL values in a column of a database table can be tested by using the operators IS NULL and IS NOT NULL. Suppose to display the records with NULL values in the Phone column of the Employee table, the following expression can be written in SQL.

```
SELECT emp_name, phone FROM Employee
WHERE phone IS NULL;
```

Constraints can be applied to table columns to prevent the addition of invalid data or deletion of data that is required to maintain the overall consistency of the database. The constraints are used to control the data being entered into a database table.

In SQL, **NOT NULL** constraint can be defined at the column level in addition with the primary and foreign key. This constraint when defined on a column means that the column cannot be left empty. It becomes a mandatory column and a value must be entered into it.

The syntax for it is as:

```
<column_name> <datatype>(<size>) NOT NULL
```

For example, to create a student table, the SQL statement is –

```
CREATE TABLE Student(
Roll_no varchar2(10),
Name varchar2(20),
D_o_b date NOT NULL,
Address varchar2(30));
```

When inserting values in the columns of the Student table, the date of birth (d_o_b) field if not entered will display a error message. This field value has to be entered or each row in the table data.

The NOT NULL constraint can be applied only at the column level of a table.

1.15 DISTINCT Clause

The SQL DISTINCT clause is used to eliminate the retrieving of the duplicate rows from a database table. It works with the SQL SELECT clause and it selects only the distinct or unique data from the database tables. For example, from the EMPLOYEE table of figure Table1, if we want to list the positions given to the employees, the SQL statement would be as :

```
SELECT DISTINCT position
FROM Employee;
```

The result of this query would be given as :

Position

Manager

Asst. Manager

AGM

Executive

The SQL DISTINCT expression returns a list of the position found in the 'position' column of the Employee table but it will remove the duplicates and will give only a single entry for each position.

1.16 Check Your Progress

Q.1.Fill up the blanks:

- a) An _____ is a symbol specifying an action that is performed on one or more expressions.
- b) Logical operators join the simple conditions in SQL to form _____.
- c) AND operator returns result only when all of the conditions given in the SQL statement are _____.
- d) The OR operator in SQL returns result only when _____ of the condition in the expression is true.
- e) To compare a value to a list of values specified in an SQL expression, the _____ operator is used.
- f) The IN operator returns _____ if the compared value matches at least one of the values in the list.

Q.2. State TRUE or FALSE:

- a) To display records which do not satisfy the conditions specified in the SQL expression, the NOT operator is used.
- b) The NOT operator cannot be used in conjunction with any other operator in SQL.
- c) While evaluating the SQL statements with multiple operators, the operators with higher precedence are evaluated first before evaluating those with lower precedence.
- d) Logical operators has higher precedence than the arithmetic operators.
- e) The LIKE, BETWEEN and IN operators are comparison operators.

- f) The LIKE operator can include two “wildcard” characters underscore (_) and percent sign (%).

Q.3.Fill in the blanks:

- a) The NOT operator can also be used with a comparison operator to _____ the result of the comparison.
- b) _____ determines the sequence in which the operations are performed in an SQL expression with multiple operators.
- c) OR operator has _____ precedence than NOT.
- d) LIKE searches for a specified _____ in a column of a database table.
- e) The wildcard character underscore (_) allows to match a _____ character.
- f) The operators with higher precedence are evaluated _____ before evaluating those with lower precedence.

1.7 Answer to Check Your Progress

Ans. to Q. No. 1 : a) Operator, b) Compound conditions, c) True,
d) any, e) IN, f) TRUE

Ans. to Q. No. 2 : a) True, b) False, c) True, d) False, e) True, f) True

Ans. to Q. No. 3 : a) negate, b) Operator precedence, c) lower
d) pattern, e) single, f) first

Unit-3

Normalization

- 1.1** Learning Objectives
- 1.2** Introduction
- 1.3** Normalization and Its Objectives
- 1.4** Normal Forms
 - 1.4.1** First Normal Form (1NF)
 - 1.4.2** Second Normal Form (2NF)
 - 1.4.3** Third Normal Form (3NF)
 - 1.4.4** Boyce-Codd Normal Form (BCNF)
 - 1.4.5** Fourth Normal Form (4NF)
 - 1.4.6** Fifth Normal Form (5NF)
- 1.5** Check Your Progress
- 1.6** Answers To Check Your Progress

1.1 Learning Objectives

After going through this unit, the learner will be able to:

- understand what is normalization
- list out the objectives of normalizations
- describe 1st, 2nd and 3rd normal forms
- illustrate fourth and fifth normal forms

1.2 Introduction

In the previous unit, we have learn about the two important concepts in database design i.e. functional dependency and decomposition which helps in minimizing the redundancy in database design. In this unit , we will concentrates on the discussion of normalization. Normalization is the process of efficiently organizing data in a database. Normalization is the name given to the process of simplifying the relationship among data elements in a record. We will introduce you the different types of normalizations and brief discussion on it.

1.3 Normalization and Its Objectives

Normalization is a process of decomposing a set of relations(table) with anomalies to produce smaller and well-structured relations that contain minimum or no redundancy. The basic objectives of normalization are to reduce redundancy, which means that information is to be stored only once. Storing information several times leads to wastage of storage space and increase in the total size of the data stored. Normalization provides the designer with a systematic and scientific process of grouping of attributes in a relation.

The normalization process can be defined as a procedure of analyzing and successive reduction of the given relational schemas based on their FDs and primary keys to achieve the desirable properties of -

- i. minimizing redundancy,
- ii. minimizing insertion, deletion and update anomalies.

The process of normalization was first proposed by E.F. Codd. Normalization is a bottom up design technique for relational database.

The objectives of the normalization process are:

1. To create a formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.
2. To free relations from undesirable insertion, update and deletion anomalies.
3. To reduce the need for restructuring the relations as new data types are introduced.
4. To carry out series of tests on individual relation schema so that the relational database can be normalized to some degree. When a test fails, the relation violating that test must be decomposed into relations that individually meet the normalization test

1.4 Normal Forms

Whenever the simple rules of functional dependencies are applied to a relations, it transforms the relations into a state which called normal form. The normal forms are used to ensure that various types of anomalies and inconsistencies are not introduced into the database. Various types of normal forms are used in relational data base, they are :

- First normal form (1 NF)
- Second normal form (2 N)
- Third normal form (3 NF)
- Boyce/Codd normal form (BCNF)

- Fourth normal form (4 NF)
- Fifth normal form (5NF)

1.4.1 First normal form (1 NF)

A relation is said to be in first normal form if the values in the domain of each attribute of the relation are atomic. The first normal form prohibits multivalued attributes, composite attributes and their combinations. It means that, 1NF disallows having a set of values, a tuple of values, or combination of both as an attribute value for a single tuple.

Let us consider the relation TRAVEL_INFO as shown in the figure.

PERSON	VISITED_CITY	
HEMANGA	CITY	JOURNEY_DATE
	DELHI	04-07-02
	MUMBAI	15-09-03
BIKASH	CITY	JOURNEY_DATE
	CHENNAI	05-01-02
	KOLKATA	13-02-03
	PUNE	21-08-04

Here, in the relation the domain VISITED_CITY is not simple. Hence, the relation is un-normalized. Now, let us combine the respective rows in VISITED_CITY with the value of the attribute PERSON and the resultant relation is shown below –

PERSON	CITY	JOURNEY_DATE
HEMANGA	DELHI	04-07-02
HEMANGA	MUMBAI	15-09-03
HEMANGA	AGRA	02-03-04
BIKASH	CHENNAI	05-01-02
BIKASH	KOLKATA	13-02-03
BIKASH	PUNE	21-08-04

Fig 8.2 Modified relation TRAVEL_INFO

let us consider another relation PATIENT_DOCTOR, which keeps the records of appointment details between patient and doctors. This relation is in 1NF. The relational table can be depicted as -

PATIENT_DOCTOR (P_NAME, DOB, D_NAME, DATE_TIME, PHONE, DURATION)

P_NAME	DOB	D_NAME	PHONE	DATE_TIME	DURATION
MUKUL	10-02-1975	Dr Deepak	0361-223470	10-01-05 10:00	15 (minutes)
RUHIT	27-01-1971	Dr Kalyan	0361-270612	10-01-05 11:00	10
JAMES	30-03-1977	Dr Dhiraj	0361-270615	10-01-05 10:30	10
JAMES	30-03-1977	Dr Dhiraj	0361-270615	10-01-05 09:00	10
RUHIT	27-01-1971	Dr Kalyan	0361-223470	10-01-05 10:15	15
MUKUL	10-02-1975	Dr Dhiraj	0361-270615	10-01-05 10:50	20
RANA	02-11-1980	Dr Kalyan	0361-270612	10-01-05 11:10	20
MUKUL	10-02-1975	Dr Deepak	0361-223470	06-02-05 16:00	15

Fig 8.3 Relation PATIENT_DOCTOR

From the relational table we have observed that a doctor cannot have two simultaneous appointments so D_NAME and DATE_TIME is a compolsite key. Similarly, a patient cannot have same time from two different doctors. Therefore, P_NAME and DATE_TIME attributes are also a candidate key.

Problems in 1 NF :

1. NF contains redundant information. In our example, PATIENT_DOCTOR relation has the following errors :
 - a. There exists redundant information in patients date of birth and phone number.
 - b. A doctor, who does not currently have an appointment with a patient, cannot be represented.
 - c. A patient, who does not currently have an appointment with a doctor, cannot be represented

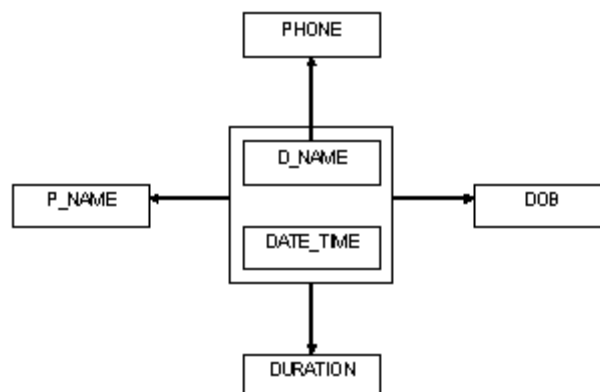


Fig 8.4 FDD of relation PATIENT_DOCTOR

Functional dependency diagram of the relation PATIENT_DOCTOR is shown. Here, P_NAME, DURATION, DOB are dependent on the key D_NAME & DATE_TIME and doctor's contact number i.e. PHONE is only dependent on the D_NAME.

1.4.2 Second Normal Form (2NF)

A relation or table is said to be in second normal form (2NF) if and only if -

- a. It is in 1NF,
- b. Each non-primary key attribute is fully functionally dependent on primary key of that relation.

2NF is an intermediate step towards higher normal forms. 2NF is based on the concept of full functional dependency. It eliminates the problems of 1NF.

So, we come to know that, no attributes of the relation (or table) should be functionally dependent on only one part of a concatenated primary key. In our example, we have seen from the functional dependency diagram that, PHONE is partially dependent only on D_NAME, for which the relation is not in 2NF.

Therefore, to bring the relation into 2NF, the information about doctor and their contact numbers have to be separated from information about patient and their appointments with doctors. Thus, the relation is decomposed into two tables, namely PATIENT_DOCTOR and DOCTOR as shown below. The relational table can be depicted as:

PATIENT_DOCTOR (P_NAME, DOB, D_NAME, DATE TIME, DURATION)
DOCTOR (D_NAME, PHONE)

P_NAME	DOB	D_NAME	DATE_TIME	DURATION
MUKUL	10-02-1975	Dr Deepak	10-01-05 10:00	15 (minutes)
RUHIT	27-01-1971	Dr Kalyan	10-01-05 11:00	10
JAMES	30-03-1977	Dr Dhiraj	10-01-05 10:30	10
JAMES	30-03-1977	Dr Dhiraj	10-01-05 09:00	10
RUHIT	27-01-1971	Dr Kalyan	10-01-05 10:15	15
MUKUL	10-02-1975	Dr Dhiraj	10-01-05 10:50	20
RANA	02-11-1980	Dr Kalyan	10-01-05 11:10	20
MUKUL	10-02-1975	Dr Deepak	06-02-05 16:00	15

Fig 8.5 Relation PATIENT_DOCTOR

D_NAME	PHONE
Dr Deepak	0361-223470
Dr Kalyan	0361-270612
Dr Dhiraj	0361-270616
Dr Dhiraj	0361-270616
Dr Kalyan	0361-223470
Dr Dhiraj	0361-270616
Dr Kalyan	0361-270612
Dr Deepak	0361-223470

Fig 8.6 Relation DOCTOR

The functional dependency diagram of the above two relations are shown below:

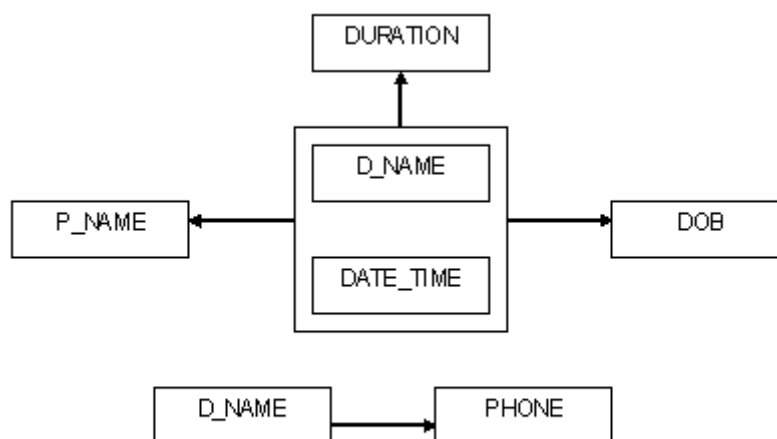


Fig 8.7 FFD of relation PATIENT_DOCTOR

Problems in 2 NF:

In the relation (or table) PATIENT_DOCTOR, deleting a record from it may lose patient's details.

Any changes made on the PATIENT_DOCTOR table may involve in changing multiple records because the information in the table is still redundant.

1.4.3 Third Normal Form (3NF)

A relation or table is said to be in third normal form (3NF) if the relation is in 2NF and the non-prime attributes are -

- a. mutually independent,
- b. Functionally dependent on the primary key.

It means that, no attributes of the relation should be transitively functionally dependent on the primary key. Thus, in 3NF, no non-prime attribute is functionally dependent on another non-prime attribute. This means that a relation in 3NF consists of the primary key and a set of independent nonprime attributes. 3NF is based on the problem of transitive dependency. The 3NF eliminates the problem of 2NF.

In our example, in the Fig 8.5, relation PATIENT_DOCTOR, there is no dependency between the attributes P_NAME and DURATION. Again, P_NAME and DOB are mutually dependent. So, the relation is not in 3NF.

To bring the relation into 3NF, it has to be decomposed and remove the attributes that are not directly dependent on the primary key. Now, using the transitive dependency, DOB can be linked to the primary key, through its dependency on the P_NAME. The functional dependency diagram is shown below. Now, the relations uses are –

PATIENT_DOCTOR(P_NAME, D_NAME, DATE_TIME, DURATION)
DOCTOR (D_NAME, PHONE)
PATIENT (P_NAME, DOB)

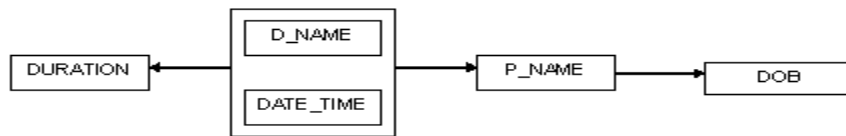


Fig 8.8 FDD for relation PATIENT_DOCTOR

Now, let us consider an another relation namely STUDENT.

STUDENT(Roll_No, Name, Dept, Year, Hostel_Name)

Roll_No	Name	Dept	Year	Hostel_Name
1784	Ruhit	Physics	1	Ganga
1648	Keshab	Chemistry	1	Ganga
1768	Gopal	Maths	2	Kaveri
1848	Raja	Botany	2	Kaveri
1682	Mrinal	Geology	3	Krishna
1485	Sumit	Zology	4	Godavari

Fig 8.9 STUDENT relation

Here, the dependency Roll_No -> Hostel_Name is transitive through the following two dependencies :

Roll_No -> Year,

Year -> Hostel_Name

Thus, the STUDENT relation is not in 3NF. To bring the relation into 3NF we can decompose the relation into two relation STUD1 and STUD2, as shown below.

STUD1(Roll_No, Name, Dept)

STUD2(Year, Hostel_Name)

Roll_No	Name	Dept	Year
1784	Ruhit	Physics	1
1648	Keshab	Chemistry	1
1768	Gopal	Maths	2
1848	Raja	Botany	2
1682	Mrinal	Geology	3
1485	Sumit	Zology	4

Fig 8.10 Relation STUD1

Year	Hostel_Name
1	Ganga
1	Ganga
2	Kaveri
2	Kaveri
3	Krishna
4	Godavari

Fig 8.11 Relation STUD2

In the above examples, the conversion into 3NF is not hard, but whenever a relation has more than one combination of attributes that may be considered as primary key then the conversion becomes problematic. Let us consider the following relation UTILIZE, shown below.

Project	Proj_Manager	Machine	Qty_Used
P1	Thomas	Excavator	5
P3	John	Shovel	2
P2	Abhishek	Drilling	5
P4	Avinash	Dumper	10
P3	John	Welding	3
P1	Thomas	Drilling	4

Fig 8.12 Relation UTILIZE

The relation stores the machines information used by both projects and project managers. Each project has one project manager and each project manager manages one project. Now it is obvious from the table that, we can consider any one of the combination of attributes as primary key, namely, {Project, Machine} or {Proj_Manager, Machine}. The FDD for relation UTILIZE is shown below.

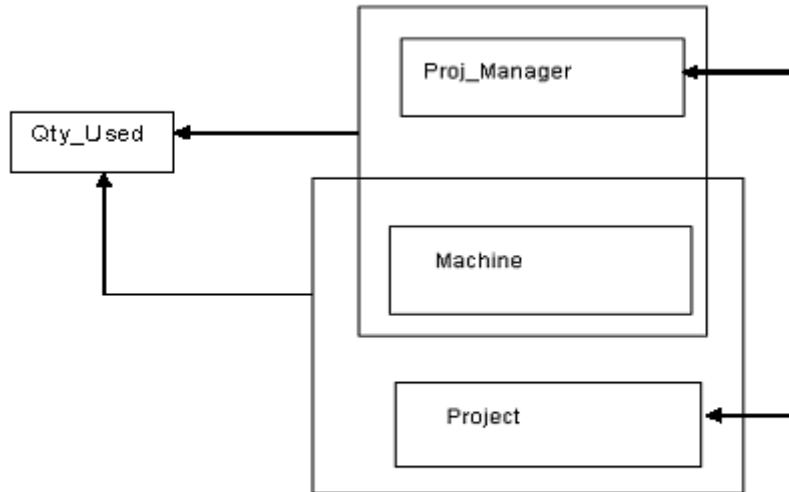


Fig 8.13 FDD of UTILIZE

In the relation, there is only one non-prime attribute called, QTY_Used, which is fully functionally dependent on each of the two relations. Thus, the relation UTILIZE is in 2NF. Moreover, there is only one non-prime attribute Qty_Used, there can be no dependencies between non-prime attributes. Thus the relation is also in 3NF.

Problems in 3NF:

If we consider the above relation i.e. UTILIZE, which is in 3NF, has the following undesirable properties:

- a. The project manager of each project is stored more than once.
- b. A project manager cannot be stored until the project has ordered some machines
- c. A project cannot be entered unless that project's manager is known.
- d. If a project's manager changes, some rows also must be changed.

1.4.4 Boyce-Codd Normal Form (BCNF)

The redundancy and the problems of 3NF can be eliminated by the use of the Boyce-Codd normal form (BCNF) which was proposed by R.F. Boyce.

A relation (or table) R is said to be in BCNF if for every nontrivial FD : $X \twoheadrightarrow Y$ between attributes X and Y holds in R. It means -

- X is super key of R,
- $X \rightarrow Y$ is a trivial FD, that is $Y \subset X$.

From the above conditions we have come to know that, a relation must only have candidate keys as determinants. Any relation in BCNF is also in 3NF and consequently in 2NF. However, a relation in 3NF is not necessarily in BCNF.

The difference between 3NF and BCNF is that - if the functional dependency $A \twoheadrightarrow B$, satisfy that B is a primary key attribute and A is not a candidate key, then 3NF will allow this dependency in a relation.

Otherwise, if the functional dependency $A \twoheadrightarrow B$, satisfy that A must be a candidate key, then this dependency will belong to BCNF.

In our example, in relation (or table) Fig 8.12, does not satisfy the condition of BCNF, as it contains the following two functional dependencies -

Proj_Manager \twoheadrightarrow Project

Project \twoheadrightarrow Proj_Manager

But neither Proj_Manager nor Project is a super key.

Now the relation UTILIZE can be decomposed into the following two BCNF relations

UTILIZE(Project, Machine, Qty_Used)

PROJECTS(Project, Proj_Manager)

Both of the above relations are in BCNF. The only FD between the UTILIZE attributes is

Project, Machine \twoheadrightarrow Qty_Used

and (Project, Machine) is a super key.

The two FDs between the PROJECTS attributes are

Project \twoheadrightarrow Proj_Manager

Proj_Manager \twoheadrightarrow Project

Both Project and Proj_Manager are super keys of relation PROJECTS and PROJECTS is in BCNF.

LET US KNOW

We have already familiar with the term multi-valued dependencies. A multi-valued dependency (MVD) is a functional dependency where the dependency may be to a set and not just a single value.

It is defined as $X \twoheadrightarrow Y$ in relation $R(X, Y, Z)$, if each X value is associated with a set of Y values in a way that does not depend on the Z values. Here, X and Y are both subsets of R. The notation $X \twoheadrightarrow Y$ is used to indicate that a set of attributes of Y shows a multi-valued dependency on a set of attributes of X.

Always remember that

1. in a relation(or table), to contain an MVD, it must have three or more attributes.
2. It is possible to have a table containing two or more attributes which are inter-dependent multi-valued facts about another attribute. For a relation to be MVD, the attributes must be independent of each other.

Let us consider the following relation to gain more concept on MVD.

Person	Project	Machine	Skill_Type
Thomas	P 1	Excavator	Analyst
John	P 1	Shovel	Programmer
Abhishek	P 2	Drilling	DBA
Abhishek	P 2	Dumper	Quality auditor
John	P 2	Welding	Programmer
Thomas	P 1	Drilling	DBA

Here, suppose that X is Person and Y is Skill_Type, then Z becomes the combination { Project, Machine }. Suppose, a particular value of Person “John” is selected. Consider all rows (tuples) that have some value of Z, for example, Project = P1 and Machine = “Shovel”. The value of Y in this tuple is ‘Programmer’.

Consider also all tuples with same value of X, that is Person, but with some other value of Z, say Project = “P2” and Machine = “Welding”. The value of Y in these tuple is again ‘Programmer’.

The same set of values of Y is obtained for Person = “John”, irrespective of the values chosen for Project and Machine. Hence, XY, or PersonSkill_Type. If we find out the possible MVD’s the following would be the results :

$Project \twoheadrightarrow Person, Skill_Type$

$Project \twoheadrightarrow Machine$

$Person \twoheadrightarrow Project, Machine$

1.4.5 Fourth Normal Form (4NF)

A table is in the fourth normal form (4NF) if it is in BCNF and does not have any independent multi-valued parts of the primary key.

The fourth normal form is related to the concept of a multi-valued dependency (MVD). In simple terms, if there are two columns - A and B - and if for a given A, there can be multiple values of B, then we say that an MVD exists between A and B.

The fourth normal form is theoretical in nature. In practice, normalization up to and including the third normal form are generally adequate. In certain situations, the designers may also have to look at the BCNF. However, rarely do we see the 4NF being employed for any real life use.

Let us consider the following STUDENT table (or relation):

S t u d e n t	S u b j e c t	L a n g u a g e
G e e t a	M y t h o l o g y	E n g l i s h
G e e t a	P s y c o l o g y	E n g l i s h
G e e t a	M y t h o l o g y	H i n d i
G e e t a	P s y c o l o g y	H i n d i
S i k h a r	G a r d e n i n g	E n g l i s h

Fig 8.14 STUDENT relation

We can see that there are two independent MVD facts in this relationship :

- a) A student can study many subjects (i.e. Student --> --> Subject)
- b) A student can learn many languages (i.e. Student -->--> Language)

The primary key for the STUDENT table is currently a composite key made up of all the three columns in the table - Student, Subject and Language. In other words, the primary key of the table is Student + Subject + Language.

The process of bringing this table into 4NF is : split the independent multi-valued components of the primary key into two tables.

Therefore, let us split these two independent multi-valued dependencies into two separate tables namely Student_Subject and Student_Language. The resulting tables are shown below:

S t u d e n t	S u b j e c t
G e e t a	M y t h o l o g y
G e e t a	P s y c o l o g y
S i k h a r	G a r d e n i n g

Fig 8.15 Relation Student_Subject

S t u d e n t	L a n g u a g e
G e e t a	E n g l i s h
G e e t a	H i n d i
S i k h a r	E n g l i s h

Fig 8.15 Relation Student_Language

We have seen that, this decomposition reduces redundancy with respect to both the independent MVD relationships, that is subject and language.

1.4.6 Fifth Normal Form (5NF)

A relation (or table) is said to be in the 5NF if and only if it is in 4NF and every join dependency in it is implied by the candidate key.

There are some relations, which cannot be decomposed into two or higher normal form relations by means of projection methods discussed in 1NF, 2NF, 3NF and BCNF. Such relations are decomposed into three or more relations, which can be reconstructed by means of a three-way or more join operation. This is called fifth normal form (5NF). The 5NF eliminates the problems of 4NF. 5NF allows for relations with join dependencies. Any relation that is in 5NF, is also in other normal forms namely 2NF, 3NF and 4NF. 5NF is mainly used from theoretical point of view and not for practical database design.

1.5 Check Your Progress

1. Select the correct answer from the following :

a) Normalization is a process of

- i.** decomposing of a set of relation.
- ii.** successive reduction of relation schema.
- iii.** deciding which attributes in a relation to be grouped together.
- iv.** all of these.

b) A normal form is

- i.** a state of a relation that results from applying simple rules regarding FD.
- ii.** the highest normal form condition that it meets.
- iii.** an indication of the degree to which it has been normalized.
- iv.** all of these.

c) In 1NF,

- i.** all domains are simple.
- ii.** in a simple domain, all elements are atomic
- iii.** both (i) & (ii)
- iv.** none of these

d) 2NF is always in

- i.** 1NF
- ii.** BCNF
- iii.** MVD
- iv.** none of these

e) A relation R is said to be in 2NF

- i.** if it is in 1NF.
- ii.** every non-prime key attributes of R is fully functionally dependent on each relation key of R.
- iii.** if it is in BCNF.
- iv.** both (i) & (ii).

f) A relation R is said to be in 3NF if the

- i.** relation R is in 2NF
- ii.** non prime attributes are mutually independent.
- iii.** functionally dependent on the primary key.
- iv.** all of these.

g) 4NF is concerned with dependencies between the elements of compound keys composed of

- i.** one attributes
- ii.** two attributes
- iii.** three or more attributes
- iv.** none of these

2. Fill in the blanks of the following :

- a) first developed the process of normalization.
- b) A relation is said to be in 1NF if the values in the domain of each attribute of the relation are
- c) 2NF can be violated only when a key is a key or one that consists of more than one
- d) In 3NF, no non-prime attribute is functionally dependent on
- e) Relation R is said to be in BCNF if for every non-trivial FD : between attributes X and Y holds in R.
- f) A relation is in BCNF if and only if every determinant is a
- g) Any relation in BCNF is also in and consequently in
- h) 4NF is violated when a relation has undesirable

1.6 Answer to Check Your Progress

1. a. (iv), b. (iv), c. (ii), d. (i), e. (iv), f. (iv), g. (iii)
2. a) E.F.Codd, b) atomic, c) composite, attribute, d) another non-prime attributes, e) X Y, f) candidate key, g) 3NF, 2NF, h) multi-valued dependencies.

Block-3

Unit-1

- 1.1 Learning Objectives
- 1.2 Introduction
- 1.3 Keys
- 1.4 Types of Keys
 - 1.4.1 Super Key
 - 1.4.2 Candidate Key
 - 1.4.3 Primary Key
 - 1.4.4 Alternate Key
 - 1.4.5 Composite Key
 - 1.4.6 Foreign Key
- 1.5. Check Your Progress
- 1.6. Answers To Check Your Progress
- 1.7. Possible Questions

1.1 Learning Objectives

After going through this unit, the learner will be able to:

- Learn about the concept of key and its uses
- Learn the different types of keys like super key, candidate key, alternate key, primary key foreign key etc.
- Define primary and foreign key in a relation
- Use composite key

1.2 Introduction

In our previous unit, we have seen that in case of relational model, the database is logically represented in the form of tables so that it can be easily understood and visualized by everyone. The roles of the keys are very important in case of relational databases. In fact, without keys relational database will not be useable at all.

In this unit, we will discuss the concept of keys in a database. The use of different types of keys will be covered in this unit.

1.3 Key

In a relational model, a database consists of relations (tables), which consists of tuples (or records/rows), which further consist of attributes (or fields/columns). We must have a way to specify how tuples within a relation are distinguished. Each relation in a relational database must have an attribute or combination of attributes such that they can uniquely identify the tuple. This unique identifier is called key. A key is that data item that exclusively identifies a record or tuple. It may consist of one or more attributes. We can split related data into different relations or tables and logically link them together with the help of keys. Without this unique identifier, there is no way to retrieve the unique tuple from a relation.

For example, let us consider the following relation (table). In this unit we may use the terminologies table, row or record and field in place of relation, tuple and attribute respectively.

STUDENT

Roll_no	Name	Marks	Grade
1	Monirupa Misra	360	A
2	Ranjita Dutta	180	C
3	Rajib Sharma	310	A
4	Kaustab Baruah	265	B
5	Apurba Bora	310	A
6	Rajib Sharma	210	B

Table 5.1

The above table gives us marks and grades of students of a particular class. There are six records in the table “STUDENT”. Each record has the following four fields: Roll_no, Name, Marks and Grade. As we can see, among the fields Name, Marks and Grade, no one field can identify a record in the table uniquely. The Name field, cannot be used as key because several student might have the same name. Marks field contains more than one same marks. Similarly, more than one students are with same Grade. So these three fields cannot be used as key. However, the field Roll_no can easily identify any row in the table uniquely. Roll numbers of students in a particular class are different. So such fields can be used as key.

1.4 Types of Key

Every key which has the property of uniqueness can be distinguished as follows:

- Super Key
- Candidate Key
- Primary Key
- Alternative Key
- Composite Key
- Foreign Key

1.4.1 Super Key

A super key is a set of columns that uniquely identifies every row in a table. For example, if there is a table STUDENT with only two columns Roll_no and Name, then the super key will be

{ Roll_no, Name }

if we assume that there are no two student in the class with the same Roll_no as well as Name.

Similarly, let us consider a EMPLOYEE table (table 5.2) consisting of the columns Emp_ID, Name and Post. We could use the Emp_ID in combination with any or all other columns of this table to uniquely identify a row in the table. Examples of superkeys in this table would be {Emp_ID}, {Emp_ID, Name} and {Emp_ID, Name, Post}.

EMPLOYEE

Emp_ID	Name	Post
001	Goutam Das	Accountant
011	Prakash Bora	Account Asstt.
023	Himangshu Das	Superintendent
033	Niharika Sarma	Financial Officer

Table 5.2

In a real database we do not need values for all of those columns to identify a row. We only need a minimal set of columns that can be used to identify a single row. In our example, the set {Emp_ID} is the minimal super key.

1.4.2 Candidate Key

A table can have more than one columns that could be chosen as the key because they individually have the capability to identify a record uniquely. These fields are termed as candidate keys. In other words, a candidate key is any set of one or more columns whose combined values are unique among all occurrences (i.e., tuples or rows or record). Since a null value is not guaranteed to be unique, no component of a candidate key is allowed to be null. Candidate keys are those attributes of a relation, which have the properties of uniqueness and irreducibility. These two properties are explained below:

Let K be a set of attributes of relation R. Then K is a candidate key for R if and only if it possesses both of the following properties:

Uniqueness: No legal value of R ever contains two distinct tuples with the same value for K.

Irreducibility: No proper subset of K has the uniqueness property.

Let us consider the following relation EMP_INFO containing some personal information of employees working in an office. Suppose all of them have passport.

EMP_INFO			
Emp_ID	Name	Passport_no	Blood Group
12341	Kunal Kashyap	M 9523421	A+
12342	Rajib Sharma	M 9515212	O+
12343	Ankur Chakraborty	M 9523123	O+
12344	Niharika Bora	F 9515456	B+
12345	Antara Dutta	F 9643521	AB+

Table: 5.3

The attribute Emp_ID and Passport_no possesses unique data item for each employee. Therefore, any of these two attribute can be chosen as the key. These two are examples of candidate keys in the above relation. The attribute Name cannot be a candidate key as more than one employee might have identical name. Similarly, several employees might have same blood group. So Blood Group cannot be chosen as key.

1.4.3 PRIMARY KEY

Every database table should have one or more columns designated as the primary key. The value this key holds should be unique for each record in the database. In a database, there can be multiple candidate keys. Out of all the available candidate keys, a database designer can identify a primary key. The primary key should be chosen such that its attributes are never or very rarely changed.

A primary key is a field or combination of fields that uniquely identify a record in a table, so that an individual record can be located without confusion. Depending on its design, a table or relation may have arbitrarily many unique keys but at most one primary key. For example, let us assume we have a table called `EMPLOYEE_ADDRESS` that contains some information for every employee in an organization. We should need to select an appropriate primary key that would uniquely identify each employee. Our first thought might be to use the employee's name i.e, `Emp_Name`. But this would not work properly because two or more employees with the same name might be possible in the organization. The `Location` field of a person cannot be chosen as primary key since it is likely to change. A better choice might be to use a unique `Emp_ID` number that the organization assign to each employee when they are appointed. `Emp_ID` can be a primary key as it does not changed till the person is working in the same organization.

EMPLOYEE_ADDRESS

PK

Emp_ID	Emp_Name	Location
1231	Gautam Baruah	GS Road Guwahati
1232	Arindam Dutta	RG Barua Road Guwahati
1233	Meghali Gogoi	Chandmari Guwahati
1234	Bornalee Sharma	Jalukbari Guwahati
1235	Arindam Dutta	Chandmari Guwahati

Table: 5.3

In the table 5.1., student's `Roll_no` would be a good choice for a primary key in the `STUDENTS` table. The student's name would not be a good choice, as there is always the chance that more than one student with same name. Some other examples of primary keys are Social Security Numbers (associated with a specific person) , `ISBN_no` (associated with a specific book).

A primary key is a special case of unique keys. Unique key constraint is used to prevent the duplication of key values within the rows of a table and allow null values. Primary key allows each row in a table to be uniquely identified and ensures that no duplicate rows exist and no

null values are entered. Thus primary key constraint can be defined as a rule that says that the primary key fields cannot be null and cannot contain duplicate data.

Once we decide upon a primary key and set it up in the database, the database management system (DBMS) will enforce the uniqueness of the key. If we try to insert a record into a table with a primary key that duplicates an existing record, the insert will fail. Sometimes, a table just does not have a primary key. In such cases, we may need to introduce an additional column which contains unique values. Most databases are also capable of generating their own primary keys. Microsoft Access, for example, may be configured to use the AutoNumber data type to assign a unique ID to each record in the table. While effective, this is a bad design practice because it leaves us with a meaningless value in each record in the table. It is better to use that space by storing some useful data.

Properties of Primary Key

To qualify as a primary key for an entity, an attribute must have the following properties:

Stable:

The value of a primary key must not change or should not become NULL throughout the life of an entity. A stable primary key helps to keep the model stable. For example, if we consider a patient record, the value for the primary key (Patient number) must not change with time as would happen with the age field.

Minimal:

The primary key should be composed of the minimum number of fields that ensures the occurrences are unique.

Definitive:

A value must exist for every record at creation time. Because an entity occurrence cannot be substantiated unless the primary key value also exists.

Accessible:

Anyone who wants to create, read or delete a record must be able to see the Primary key value.

1.4.4 Alternate key

As we have seen, it is possible for a relation to have two or more candidate keys. If we chose any one of them as primary key, then the remaining keys will be termed as alternate key. The alternate key (or secondary key) is any candidate key which is not selected to be the primary key. For the illustration of alternate key, let us consider the following table ELEMENT which stores some information like element name, symbol, atomic number of the elements of periodic table.

Name	Symbol	Atomic_no
Hydrogen	H	1
Helium	He	2
Lithium	Li	3
Berelium	Be	4
Boron	B	5
Carbon	C	6
Nitrogen	N	7
Oxygen	O	8
Flurine	F	9
Nion	Ni	10

Table: 5.4

All the three fields can individually identify each element in the table. So any of these three fields can be chosen as the primary key . If we choose Symbol as the primary key; Name and Atomic_no would then be alternate keys. Similarly, in the EMP_INFO (table 5.3), if we consider Emp_ID as the primary key then Passport_no will be the alternate key.

1.4.5 Composite key

In some situations, while designing a database, there may not be a particular column or field that can individually identify a record uniquely in a table. In such cases, we may require to select two or more fields so that combination of those can identify each record uniquely. These combination of fields is known as composite key. It is used when a record cannot be uniquely identified by a single field.

For the illustration of composite key, let us consider the following table ITEM with the fields Supplier_ID, Item_ID, Item_Name and Quantity. This table gives us the information which supplier sells

which item. As we can see, any of these fields individually cannot identify a row in the table uniquely. But if we combine Supplier_ID and Item_ID, then these together can easily identify any row in the table uniquely. Thus, Supplier_ID and Item_ID together becomes a composite key.

Composite Key

ITEM

Supplier_ID	Item_ID	Item_Name	Quantity
S ₁	I ₁	AC	5
S ₁	I ₂	Inverter	8
S ₂	I ₂	Inverter	4
S ₂	I ₃	UPS	15
S ₂	I ₄	Generator	5
S ₃	I ₃	UPS	10

Table: 5.5

1.4.6 Foreign keys

One important type of key that we will discuss in this unit is the foreign key. These keys are used to create relationships between tables.

A foreign key is a field in a relational table that matches the primary key column of another table. It identifies a column or a set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table. The columns in the referencing table must be the primary key or other candidate key in the referenced table. The values in one row of the referencing columns must occur in a single row in the referenced table. Thus, a row in the referencing table cannot contain values that do not exist in the referenced table. This way references can be made to link information together and it is an essential part of database normalization. Multiple rows in the referencing table may refer to the same row in the referenced table.

For example in an employees database, let us imagine that we wanted to add a table DEPARTMENT containing departmental information to the database. We would also want to include information about the employees in the department, but it would be redundant to have the same information in two tables (EMPLOYEE and DEPARTMENT). Instead, we can create a relationship between the two tables.

EMPLOYEE

PK Emp_ID	Name	Post	FK Department_Name
A01	Goutam Bora	Accountant	Sales
A02	Manash Saikia	Manager	Marketing
A03	Himangshu Das	Financial Officer	Human Resource
A04	Niharika Baruah	Accountant	Production
A05	Pranjal Hazarika	Law Officer	Human Resource
A06	Meghna Roy	Receptionist	Executive Sales
A07	Diganata Bora	Manager	Production
A08	Smita Roy	Law Officer	Production
A09	Barnalee Sharma	Receptionist	Marketing
A10	Dhrub Sharma	Manager	Human Resource
A11	Kamalesh Das	Manager	Sales

Table: 5.6

DEPARTMENT

PK Department_Name	Manager	Dept_Code
Human Resource	Dhruba Sharma	D1
Marketing	Manash Saikia	D2
Production	Diganata Bora	D3
Sales	Kamalesh Das	D4

Table: 5.7

Let us assume that the DEPARTMENT table uses the Department_Name column as the primary key. To create a relationship between the two tables, we add a new column to the EMPLOYEE table called Department_Name. We then fill in the name of the department to which each employee belongs. The Department_Name column in the EMPLOYEE table is a foreign key (FK) that references the DEPARTMENT table. The database will then enforce referential integrity by ensuring that all of the values in the Department column of the Employees table have corresponding entries in the DEPARTMENT table.

Again, let us consider a book database. The BOOKS table has a link to the publishers table. The Pub_ID column is the primary key for the PUBLISHERS table and ISBN_no is the primary key for the BOOKS table. The BOOKS table also contains a Pub_ID column which matches the primary key column of the publishers table. This Pub_ID is the foreign key in the BOOKS table. The Pub_ID field in the BOOKS table indicates which publisher a book belongs to.

PUBLISHERS

PK

Pub_ID	Pub_Name	City	State
--------	----------	------	-------

Table: 5.8

BOOKS

PK

ISBN_no	Book_name	Author_name	Pub_ID	Price	Pub_date
---------	-----------	-------------	--------	-------	----------

FK

Table:5.9

Although the primary purpose of a foreign key constraint is to control the data that can be stored in the foreign key table, it also controls changes to data in the primary key table. For example, if the row for a publisher is deleted from the publishers table, and the publisher's ID is used for books in the BOOKS table, the relational integrity between the two tables is broken; the deleted publisher's books are orphaned in the BOOKS table without a link to the data in the publishers table. A foreign key constraint prevents this situation. The constraint enforces referential integrity by ensuring that changes cannot be made to data in the primary key table if those changes invalidate the link to data in the foreign key table. If an attempt is made to delete the row in a primary key table or to change a primary key value, the action will fail if the deleted or changed primary key value corresponds to a value in the foreign key constraint of another table. To change or delete a row in a foreign key constraint successfully, we must first either delete the foreign key data in the foreign key table or change the foreign key data in the foreign key table, thereby linking the foreign key to different primary key data. i.e., a primary key constraint cannot be deleted if referenced by a foreign key constraint in another table; the foreign key constraint must be deleted first.

1.5 Check Your Progress

1. State whether the following statements are true or false:

- (a) A key is that data item that exclusively identifies a record.
- (b) A table or relation may have arbitrarily many unique keys but at most one primary key.
- (c) The alternate key is any candidate key which is not selected to be the primary key.
- (d) Unique key constraint is used to allow the duplication of key values within the rows of a table and allow null values.
- (e) The primary key fields cannot be null and cannot contain duplicate data.

2. State whether the following statements are True or False:
 - (a) In a relational database, the foreign key of a relation would be the primary key of an another relation.
 - (b) Foreign Key represents relationship between the tables.
 - (c) There may be more than one foreign key in a table.

1.6 Answer to Check Your Progress

1. (a) True (b) True (c) True (d) False (e) True
2. (a) True , (b) True, (c) False

1.7 Possible Questions

1. Explain the following keys with examples:
 - (i) Primary Key (ii) Candidate Key (iii) Super Key (iv) Alternate Key (v) Foreign Key
2. Create a relation student and mark the different types of keys in it.
3. Explain the significance of primary key and foreign key in employee database.
4. What is Keys? What are the different types of Keys?
5. Difference between candidate key and Super key?
6. Discuss the properties of primary key?

Unit-2

- 1.1 Learning Objectives
- 1.2 Introduction
- 1.3 Back up of Database
 - 1.3.1 SQL Server Backup – Scopes and Types
 - 1.3.2 Backup Scopes
 - 1.3.3 Backup Types
 - 1.3.4 Back Up Tools
- 1.4 Types of Database Failure
- 1.5 Types of Database Recovery
 - 1.5.1 Developing a Backup and Recovery Strategy
 - 1.5.2 General Types of Recovery
 - 1.5.3 Structure of Recovery
 - 1.5.4 Recoverable Database Backup Operations
 - 1.5.5 Automated Backup Operations
- 1.6 Database Security
- 1.7 Check Your Progress
- 1.8 Answers to Check Your Progress
- 1.9 Possible Question

1.1 Learning Objectives

After going through this unit, the learner will be able to:

- understand the importance of back up in a database
- know the recovery process
- learn the different types of recovery operations
- difference between backup and recovery process

1.2 Introduction

Although most database systems have incorporated backup and recovery tools into their interfaces and infrastructure with the growing dependency in the workplace on information and general, and the information in a database specifically, there has never been a time when safe backups and reliable recoveries were more important. It is not just the data files that need to be part of the backup process. You must also backup the transaction logs of the database as well. Without the transaction logs the data files are useless in a recovery event. How often

you choose to perform these backup routines is really dependent on the data requirements of a company.

1.3 Back up of Database

Backing up of a database is only the first step in the process. The next step is to make sure those backups are protected. You also need to test the backups and to ensure that they can be used to restore your database. Probably the most common database backup technique involves backing up the database to a disk on the same server. This is fine, provided the disk is a separate physical RAID array from the one that your database sits on. Since backups are used to recover from worst-case scenarios, they need to be protected from such disasters. After all, what good are backups that will just be lost when the server fails?

The two most common things to do with these backups are to either back them up to tape shortly after they are saved to disk or move them to another server for long-term storage. Either solution is acceptable since you are left with a secondary backup. This way if the server fails and takes your original backups with it, you can still restore the database.

Once you have your backups saved to another location — either tape or another server — you then must ask yourself whether those backup can be restored. Ensuring that your backups are going to be useful is an important step in the backup process. Using the Verify Backup Integrity checkbox in the maintenance plan is not enough; because it simply makes sure that the header of the backup is correct without verifying the validity of the backup.

1.3.1 SQL Server Backup – Scopes and Types

One of the major advantages that enterprise-class databases offer over their desktop counterparts is a robust backup and recovery feature set. Microsoft SQL Server provides database administrators with the ability to customize a database backup and recovery plan to the business and technical requirements of an organization.

In this unit, we will explore the process of backing up data with Microsoft SQL Server. When you create a backup plan, you'll need to create an appropriate mix of backups with varying backup scopes and backup types that meet the recovery objectives of your organization and are suitable for your technical environment.

1.3.2 Backup Scopes

The scope of a backup defines the portion of the database covered by the backup. It identifies the database, file(s) and/or file group(s) that SQL Server will backup. There are three different types of backup scope available in Microsoft SQL Server:

- **Database backups** cover the entire database including all structural schema information, the entire data contents of the database and any portion of the transaction log necessary to restore the database from scratch to its state at the time of the backup. Database backups are the simplest way to restore your data in the event of a disaster, but they consume a large amount of disk space and time to complete.
- **Partial backups** are a good alternative to database backups for very large databases that contain significant quantities of read-only data. If you have read-only file groups in your database, it probably doesn't make sense to back them up frequently, as they do not change. Therefore, the scope of a partial backup includes all files in the primary file group, all read/write file groups, and any read-only file groups that you explicitly specify.
- **File backups** allow you to individually backup files and/or file groups from your database. They may be used to complement partial backups by creating one-time-only backups of your read-only file groups. They may also play a role in complex backup models.

1.3.3 Backup Types

The second decision you need to make when planning a SQL Server database backup model is the type of each backup included in your plan. The backup type describes the temporal coverage of the database backup. SQL Server supports two different backup types:

- **Full Backups** include all data within the backup scope. For example, a full database backup will include all data in the database, regardless of when it was last created or modified. Similarly, a full partial backup will include the entire contents of every file and file group within the scope of that partial backup.
- **Differential Backups** include only that portion of the data that has changed since the last full backup. For example, if you perform a full database backup on Monday

morning and then perform a differential database backup on Monday evening, the differential backup will be a much smaller file (that takes much less time to create) that includes only the data changed during the day on Monday.

1.3.4 Backup Tools

You should keep in mind that the scope and type of a backup are two independent decisions made when creating your backup plan. As described above, each type and scope allows you to customize the amount of data included in the backup and, therefore, the amount of time required to backup and restores the database in the event of a disaster.



Using the SQL Database Backup and Restore console agent we can automate the backup of SQL Databases. The input to the agent is in the form of an ini file making it fully interactive. The agent can write event logs. Using this agent we can create compressed backup of SQL Databases, restore the SQL Database using this backup and delete the intermediate file. For best results this agent should be used with Mobility backup software in the client server mode Other Tools of database backup are

- Active@ Partition Recovery
- Memory Card Recovery Software
- Acronis True Image Home Upgrade
- R-Drive Image
- Paragon Drive Backup Personal
- Recover USB Drive Files
- BootMaster Rescue Disk for Windows
- Driver Magician
- Handy Recovery
- WordFIX Data Recovery
- USB External Drive Recovery

1.4 Types of Database Failure

Database failures can be classified as transaction failure, media failure and system failures. Some of the cause for which transaction of a database to fail in the middle of execution:

1. **System crash or computer failure:** A hardware, software or network failure or error may occurs in the computer system during transaction execution.
2. **Media failures:** hardware crashes are generally media failures, such as main memory failure.
3. **A transaction or system error:** Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. It may be occurred due to some logical error.
4. **Concurrency control enforcement:** The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock.
5. **Disk Failure:** Some of the disk may be lose their data because of read or write malfunction or because of a disk read/write head crash.
6. **Physical problems:** This referred to an endless list of problems that include power or air conditioning failure, fire, theft, overwriting disk or tapes by mistakes, mounting of a wrong tape by the operator.

1.5 Types of Database Recovery

The types of database recovery are

- Developing A Backup and Recovery Strategy
- General Types of Recovery
- Structure of Recovery
- Recoverable Database Backup Operations
- Automated Backup Operations

1.5.1 Developing A Backup and Recovery Strategy

Whether a business is small, medium or large business, it must have a well-written plan for backing up the servers. Planning a backup strategy up-front and documenting not only the backup process but also the restore process, will save you a ton of time in the end. Because of its value to the company and the sensitive nature of it, the classification of data must be carefully considered in the planning stage. Based upon these classifications, the backup and restore plan will need to be tested and adjusted. While planning stage, data should be ranked according to sensitivity and value to the business.

With data that is highly valuable to a company, plans must include an increased backup frequency due to the nature of the costs incurred while recapturing data in case of a to disaster. Recoverability plans must also consider the availability requirements of this data. With highly sensitive data, plans must include encryption of backups, especially when this data is stored offsite.

With a SQL Server, DBA's should also be concerned with the OS, the applications that the server runs and finally the databases. In other words, the entire server needs a backup and recovery plan. User databases are critical to backup plan, but system databases that contain significant information like Users, SQL Jobs and other system functionality, must also be taken into account

A database can become unusable because of hardware or software failure, or both. You may, at one time or another, encounter storage problems, power interruptions, or application failures, and each failure scenario requires a different recovery action. Protect your data against the possibility of loss by having a well-rehearsed recovery strategy in place. Some of the questions that you should answer when developing your recovery strategy are:

1. Will the database be recoverable?
2. How much time can be spent recovering the database?
3. How much time will pass between backup operations?
4. How much storage space can be allocated for backup copies and archived logs?
5. Will table space level backups be sufficient, or will full database backups be necessary?

A database recovery strategy should ensure that all information is available when it is required for database recovery. It should include a regular schedule for taking database backups and, in the case of partitioned database systems, include backups when the system is scaled (when database partition servers or nodes are added or dropped). Your overall strategy

should also include procedures for recovering command scripts, applications, user-defined functions, stored procedure codes.

The concept of a database backup is the same as any other data backup: taking a copy of the data and then storing it on a different medium in case of failure or damage to the original. The simplest case of a backup involves shutting down the database to ensure that no further transactions occur, and then simply backing it up. You can then rebuild the database if it becomes damaged or corrupted in some way.

The rebuilding of the database is called recovery. Version recovery is the restoration of a previous version of the database, using an image that was created during a backup operation. Roll forward recovery is the reapplication of transactions recorded in the database log files after a database or a table space backup image has been restored.

1.5.2 General Types of Recovery

Crash recovery is the automatic recovery of the database if a failure occurs before all of the changes that are part of one or more units of work (transactions) are completed and committed. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred.

Recovery log files and the recovery history file are created automatically when a database is created (Figure 1). These log files are important if you need to recover data that is lost or damaged.

Each database includes recovery logs, which are used to recover from application or system errors. In combination with the database backups, they are used to recover the consistency of the database right up to the point in time when the error occurred.

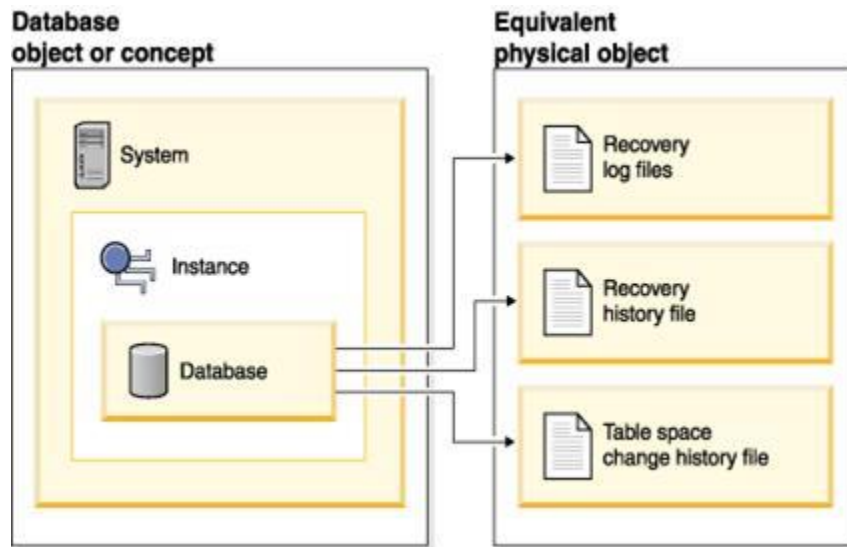
The recovery history file contains a summary of the backup information that can be used to determine recovery options, if all or part of the database must be recovered to a given point in time. It is used to track recovery-related events such as backup and restore operations, among others. This file is located in the database directory.

The table space change history file, which is also located in the database directory, contains information that can be used to determine which log files are required for the recovery of a particular table space.

1.5.3 Structure of Recovery

You cannot directly modify the recovery history file or the table space change history file; however, you can delete entries from the files using the PRUNE HISTORY command. You can also use the `rec_his_reten` in database configuration parameter to specify the number of days that these history files will be retained.

Figure 1. Database recovery files



Data that is easily recreated can be stored in a non-recoverable database. This includes data from an outside source that is used for read-only applications, and tables that are not often updated, for which the small amount of logging does not justify the added complexity of managing log files and rolling forward after a restore operation. Non-recoverable databases have the `logarchmeth1` and `logarchmeth2` database configuration parameters set to “OFF”. This means that the only logs that are kept are those required for crash recovery. These logs are known as active logs, and they contain current transaction data. Version recovery using offline backups is the primary means of recovery for a non-recoverable database. (An offline backup means that no other application can use the database when the backup operation is in progress.) Such a database can only be restored offline. It is restored to the state it was in when the backup image was taken and roll forward recovery is not supported.

Data that cannot be easily recreated should be stored in a recoverable database. This includes data whose source is destroyed after the data is loaded, data that is manually entered into tables, and data that is modified by application programs or users after it is loaded into the database. Recoverable databases have the `logarchmeth1` or `logarchmeth2` database configuration parameters set to a value other than “OFF”. Active logs are still available for

crash recovery, but you also have the archived logs, which contain committed transaction data. Such a database can only be restored offline. It is restored to the state it was in when the backup image was taken. However, with roll forward recovery, you can roll the database forward (that is, past the time when the backup image was taken) by using the active and archived logs to either a specific point in time, or to the end of the active logs.

1.5.4 Recoverable Database Backup Operations

Recoverable database backup operations can be performed either offline or online (online meaning that other applications can connect to the database during the backup operation). Online table space restore and roll forward operations are supported only if the database is recoverable. If the database is non-recoverable, database restore and roll forward operations must be performed offline. During an online backup operation, roll forward recovery ensures that all table changes are captured and reapplied if that backup is restored.

If you have a recoverable database, you can back up, restore, and roll individual table spaces forward, rather than the entire database. When you back up a table space online, it is still available for use, and simultaneous updates are recorded in the logs. When you perform an online restore or roll forward operation on a table space, the table space itself is not available for use until the operation completes, but users are not prevented from accessing tables in other table spaces.

1.5.5 Automated Backup Operations

Since it can be time-consuming to determine whether and when to run maintenance activities, such as backup operations, you can use the Configure Automatic Maintenance wizard to do this for you. With automatic maintenance, you specify your maintenance objectives, including when automatic maintenance can run. DB2 then uses these objectives to determine if the maintenance activities need to be done and then runs only the required maintenance activities during the next available maintenance window (a user-defined time period for the running of automatic maintenance activities).

Note: You can still perform manual backup operations when automatic maintains is configured. DB2 will only perform automatic backup operations if they are required.

1.6 Database Security

Database security protects the database against the unauthorized persons to access a certain part of a database or the whole database. Database security is very broad area that addresses many issues.

Types of Security:

1. In database Management System Some information may be deemed to be private, such that it cannot be accessed by any unauthorized user.
2. Database security issues some policy for accessing data in various levels like Governmental, Institutional, and corporate level as to what kind of information should not be made publicly available.
3. System related security.
4. In case of database security, an organization should maintain security at various levels and to categorize the data and the users' base on these classifications. As for example – A Super admin level users have the authority to insert, read and writes operation on the data. In admin level user have the authority to insert and read the data but these type of user cannot delete or modify the data in database permanently and in general user level they can only read the data from database server

Goals of database security:

1. Loss of integrity: integrity is lost if unauthorized changes are made to the data by intentionally or accidentally acts. If the lost of the system or data integrity is not corrected, continued use of the corrupted data could result in inaccuracy, fraud or erroneous decisions.
2. Loss of availability,
3. Laws of confidentiality- database confidentiality refers to the protection of data from unauthorized users. Unauthorized, unanticipated or unintentional disclosure could result in loss of public confidence, embarrassment, or legal action against the organization.

1.7 Check Your Progress

Find True/False from the following :

1. Back Up and recovery is same thing.
2. Manual recovery is not possible.
3. SQL Server support back up but not the recovery process.

4. Back up and recovery are inherent process of database software.
5. Recovery gets more importance than Back up.
6. Normalization reduces the back up storage space.

1.8 Answer to Check Your Progress

1. False 2. False 3. False
4. True 5. False 6. True

1.9 Possible questions

1. Try to make difference between database Recovery and Back up process.
2. Describe the various type of back up operations.
3. Describe the various type of recovery process.
4. What is the database administrator responsibility in terms of recovery operations?
5. What are the steps to restore the database in SQL Server?
6. What is database security?
7. What are the causes of database failure?

Unit-3

- 1.1 Learning Objectives
- 1.2 Introduction
- 1.3 Model Concept
 - 1.3.1 Basic Terminology
 - 1.3.2 Relational Schema and Instances
- 1.4 Integrity Constraints
 - 1.4.1 Entity Integrity Constraints
 - 1.4.2 Referential Integrity Constraints
- 1.5 Domain Constraints
- 1.6 The CODD Commandments
- 1.7 Check Your Progress
- 1.8 Answers to Check Your Progress
- 1.9 Possible Question

1.1 Learning Objectives

After going through this unit, the learner will be able to learn:

- Describe relational model and its advantages
- State different integrity constraints.
- Describe how data are organized in the form of tables.

1.2 Introduction

In the previous unit, we have discussed the properties of basic and commercial data models and details of Entity-Relationship model.

This unit is an attempt to provide you the concept of relational model. Most of the commercial DBMS products available in the industry are relational at core. In this unit we will discuss the terminology, operators and operations used in relational model. There are certain restrictions in formulating the relation table. Those restrictions will also be discussed in this unit.

1.3 Model Concept

A model in database system basically defines the structure or organization of data and a set of operations on that data. Relational model is a simple model in which database is represented as a collection of “Relations”, where each relation is represented by a two dimensional table. Thus, because of simplicity, it is most commonly used in real world. Following table represents a simple relation:

R_NO	S_NAME	ADDRESS	MARKS
10	Sanjib Kaur	Block -4, Noonmati	69
12	Padip Sen	Ganeshguri	75
15	Bipul Prasad	Bamunimaidan	58

Figure 4.1 A sample STUDENT relation

Properties of a table

A table should contain the following properties:

- a. Each entry in a table represent one data item and two column headings with the same name are not allowed.
- b. In each column, the data items are of the same data type.
- c. Each column is assigned with a distinct heading.
- d. All rows are distinct; duplicate rows are not allowed.
- e. Both the rows and the columns can be viewed in any sequence at any time without affecting the information.

A relational database model uses a collection of tables to represent both data and the relationships among those data items. Each table has multiple columns, and each column has a unique name.

Properties of RDBMS

A relational database management system (RDBMS) has the following properties:

- a. Stores data in the form of tables.
- b. Does not require the user to understand its physical implementation.
- c. Provides information about its contents and structure in the system table.
- d. Supports the concept of NULL values.

Following are some of the advantages of relational model:

- a. **Ease to use:** Convenient to define and query the database as tables contains rows and columns is quite natural even for first time users.
- b. **Flexibility:** Data can manipulate easily with the help of relational operations.
- c. **Precise:** Since use mathematical operations, it can ensure accuracy and less of ambiguity as compared to other model.
- d. **Security:** security control and authorization can also be implemented more easily. It has user's own authorization controls.
- e. **Data Independence:** Data independence is achieved more easily with normalization structure used in a relational database.
- f. **Data Manipulation Language:** The possibility of responding to query on relational algebra and relational calculus is easy in the relational database approach.

1.3.1 Basic Terminology

The basic terminologies used in relational database model are:

Tuple (record) and attribute:

Each row in a table is called a tuple and a column name is called an attribute. For example, Figure 4.3.1 represents a STUDENT relation where ROLL NO, NAME, ADDRESS and MARKS are attributes and each entry against these attributes is called tuple of relation STUDENT.

Domain:

A domain is a collection of all possible values from which the values for a given column or attribute is drawn. So, every attribute in a table has a specific domain. Values to these attributes cannot be assigned outside their domains. For example, the domain of attribute NAME is the set of all alphabetic string of finite length and the domain of a MRKS attribute should not be greater than 100 for the relation STUDENT in Figure 4.1.

Relation:

The table with all tuples and attributes is called relation. It has three components: Name that represent by the title of the relation, Degree, the number of column associated with the table and the Cardinality, the number of rows in the table. For example, Figure 4.3.1 represents a relation named STUDENT of degree 4, because it has total four attributes, and the cardinality for this relation is 3(number of rows).

1.3.2 Relational Schema and Instances

A relation consists of Relational Schema and Relation Instances.

Relational Schema: A schema specifies the relation name, its attributes and domain of each attribute. If R is the name of a relation and A1, A2, A3.....An are the attributes of R, then R(A1, A2,.....An) is called a relational schema. Each attribute takes values from a specific domain D. For example, for Figure 5.1, the relational schema is

STUDENT (R_NO, S_NAME, ADDRESS, MARKS)

where, STUDENT is the name of the Relation and R_NO, S_NAME, ADDRESS, and MARKS are four different attributes that represent the relational schema.

Relation Instance: A relation instance denoted as r is a collection of tuples for a given relational schema. The relation state of the relational schema R(A1,A2,.....An) is denoted as r(R) is a set of n-tuples.

The relation schema is also called ‘intension’ and relation state is called ‘extension’.

1.4 Integrity Constraints

The term integrity refers to the accuracy or correctness of data in the database schema and is expected to hold on every database instance of that schema. Relational model includes two general integrity constraints. They are:

1.4.1 Entity Integrity Constraints

Entity Integrity Constraints states that no primary key value can be NULL. This is because we use the primary key value to identify individual tuples in a relation. It ensures that instances of the entities are distinguishable i.e., they must have a unique identification of some kind. Primary keys perform that unique identification function in a relational database.

1.4.2 Referential Integrity Constraints

Referential Integrity Constraint is specified between two relations and is used to maintain the consistency among tuples of the two relations (not necessarily be distinct). It uses a concept of foreign key which will be explained more details in the next unit. Informally, it states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. Considering the following relations,

EMPLOYEE

p.k

ENO	ENAME	DNO
101	Robert	10
102	Smith	12
103	Robindra	12
104	John	10

DEPARTMENT

p.k

DNO	DNAME	LOCATION
10	Comp. Sc.	Jalukbari
12	Electronic Sc.	Guwahati

Figure 4.2: relational database table showing referential integrity

In the above figure EMPLOYEE and DEPARTMENT are two relations where ENO and DNO are primary keys respectively. Here the attribute DNO of EMPLOYEE table is a foreign key that gives the department number for which each employee works. Hence its value in each EMPLOYEE tuple must match the DNO value of some tuple in the DEPARTMENT relation.

1.5 Domain Constraints

It specifies that each attribute in a relation must contain an atomic value only from the corresponding domains. The data types for commercial RDBMS domains are:

- Standard numeric data types for integer
- Real numbers
- Characters
- Fixed length and variable length strings

Thus, domain constraint specifies the condition that we want to put on each instance of the relation. So, the values that appear in each column must be drawn from the domain associated with that column.

1.6 The CODD Commandments

There are twelve (12) rules formulated by E.F. Codd for RDBMS in 1970 to define the requirements more rigorously within a single product. In reality it is true to say that they do not all carry the same degree of importance, but can be obtained a good result if an RDBMS satisfies all these twelve rules. The rules are:

Rule 1: The information rule

All information is explicitly and logically represented in exactly one way – by data values in tables. In simple terms this means that if an item of data does not reside somewhere in a table in the database then it does not exist and this should be extended to the point where even such information as table, view and column names to mention just a few, should be contained somewhere in table form.

Rule 2: The rule of guaranteed access

Every item of data must be logically addressable by resorting to a combination of table name, primary key value and column name. For a table like storage structure, this rule says that at the insertion of a column and row it is necessarily find one value of a data item or null.

Rule 3: Systematic treatment of null values

In DBMS NULL values are supported in the representation of missing and inapplicable information. This support for null values must be consistent throughout the DBMS and independent of all data type.

Rule 4: Database description rule

The description of the database is held and maintained using the same logical structures used to define the data, thus allowing users with appropriate authority to query such information in the same way and using the same languages as they would any other data in the database. It implies that there must be a data dictionary within the RDBMS that is constructed of tables and/or views that can be examined using SQL. Therefore a dictionary is mandatory for RDBMS.

Rule 5: Comprehensive sub-language rule

There must be at least one language whose statements can be expressed as character strings conforming to some well-defined syntax. In real terms, the RDBMS must be completely manageable through its own extension of SQL.

Rule 6: View updating rule

All views that can be defined using combination of base tables, and theoretically updatable, must also be capable of being updated by the system. This is quite a difficult rule to interpret and with all sorts of aggregates and virtual columns, it is obviously not possible to update through some of them.

Rule 7: Insert and update rule

An RDBMS do more than just be able to retrieve relational data sets. It has to be capable of inserting, updating and deleting data as a relational set.

Rule 8: Physical independence rule

User access to the database, via monitors and application programs, must remain logically consistent whenever changes to the storage representation, or access methods to the data, are changed. For example, if an index is built and destroyed by the DBA on a table, any user should still retrieve the same data from that table.

Rule 9: Logical data independence

Application programs must be independent of changes made to the base tables. This allows many types of database design change to be made dynamically, without users being aware of them.

Rule 10: Integrity rule

The relational model includes two general integrity rules which we have discussed in already in this unit. These integrity rule implicitly or explicitly define the set of consistent database states, or changes of state, or both. Other integrity constraints can be specified during database design.

Rule 11: Distribution rule

An RDBMS must have distribution independence. Thus, RDBMS package must make it possible for the database to be distributed across multiple computers even though they are having heterogeneous platforms both for hardware and operating system.

This is one of the most attractive aspects of RDBMSs, database system built on the relational framework are well suited to today's client/server database design.

Rule 12: No subversion rule

If an RDBMS supports a lower level language that permits for example, row at-a-time processing, then this language must not be able to bypass any integrity rules or constraints of the relational language.

1.7 Check Your Progress

1. Fill in the blanks for the following.

1. In a relational database model, the columns of a table are called _____.
2. _____ is the rows in a table.
3. _____ is a collection of related files.
4. The number of columns in a table is the _____ of the relation.
5. _____ is the number of rows in a table.

6. An entity is an object that is distinguishable from other objects by a specific set of _____
7. Collection of tuples for a relational schema at a particular time is called _____
8. A relation state of a relational schema R denotes as _____ if r is the collection of tuples.
9. A relation state is also called _____
10. Relational schema contains name and _____ of that relation.

2. Fill in the blanks for the following

- a. In a relational database, _____ are not allowed to have null values.
- b. In a relational database, a referential integrity constraint is specified with the help of _____.
- c. The _____ in a record is a unique data item.
- d. In a relation, each _____ name must have different meaning.
- e. According to integrity rule 1, two _____ should be distinguishable from each other.

3. Write whether the following statements are true or false or False (T/F):

- a) Prime attribute have unique identifier.
- b) The key field may be NULL in relational database.
- c) Foreign key concept is used to explain Integrity Rule 1.
- d) Domain constraint specifies the values related to instances.
- e) According to referential integrity constraints, values of foreign key field of that table are dependent on the values of primary key field of another table.

1.8 Answer to Check Your Progress

1.
 1. Attribute 2. Tuple 3. Database 4. Degree
 5. Cardinality 6. Attribute 7. Relation instance
 8. $r(R)$ 9. Extension 10. attribute
2.
 1. (a) primary key (b) foreign key (c) key field
 - (d) attribute (e) entities
3.
 - (a) T (b) F (c) F (d) T (e) T

1.9 Possible Question

1. Define Relation and Domain as applicable to RDBMS.
2. Explain the properties adopted by a Relational Database Management System?
3. What are the advantages of Relational Model?
4. State and explain the basic components of a Relation.
5. What do you mean by Integrity Constrains? Give a brief note on it.
6. What is Domain Constraint?
7. Explain any 6 Codd's rules for RDBMS.
8. According to Codd, how NULL values of a relation can be treated in an RDBMS package.