# DBMS Full notes

database management system (Gretsa University)

# MINISTRY OF EDUCATION

## DIPLOMA IN INFORMATION COMMUNICATION TECHNOLOGY

**KENYA INSTITUTE OF CURRICULUM DEVELOPMENT**
**STUDY NOTES**

---

# Database Management System

# [DBMS]

# MODULE II: SUBJECT NO 5

---

# Contents

# CHAPTER 1: INTRODUCTION TO DATABASE MANAGEMENT SYSTEMS

## Introduction to Database management system (DBMS)

Database management system (DBMS) are computer software applications that interact with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases.

It's a set of software programs that controls the organization, storage and retrieval of data (fields, records and files) in a database. It also controls the security and integrity of the database.

The following are main examples of database applications:

• Computerized library systems

• Automated teller machines

• Flight reservation systems

• Computerized parts inventory systems

## Advantages of DBMS

The database management system has a number of advantages as compared to traditional computer file-based processing approach. The DBA must keep in mind these benefits or capabilities during databases and monitoring the DBMS.
The Main advantages of DBMS are described below.

- Controlling Data Redundancy - In non-database systems each application program has its own private files. In this case, the duplicated copies of the same data is created in many places. In DBMS, all data of an organization is integrated into a single database file. The data is recorded in only one place in the database and it is not duplicated.

- Sharing of Data - In DBMS, data can be shared by authorized users of the organization. The database administrator manages the data and gives rights to users to access the data. Many users can be authorized to access the same piece of information simultaneously. The remote users can also share same data. Similarly, the data of same database can be shared between different application programs.

- Data Consistency - By controlling the data redundancy, the data consistency is obtained. If a data item appears only once, any update to its value has to be performed only once and the updated value is immediately available to all users. If the DBMS has controlled redundancy, the database system enforces consistency.

- Integration of Data - In Database management system, data in database is stored in tables. A single database contains multiple tables and relationships can be created between tables (or associated data entities). This makes easy to retrieve and update data.

- Integration Constraints - Integrity constraints or consistency rules can be applied to database so that the correct data can be entered into database. The constraints may be applied to data item within a single record or the may be applied to relationships between records.

- Data Security - Form is very important object of DBMS. You can create forms very easily and quickly in DBMS. Once a form is created, it can be used many times and it can be modified very easily. The created forms are also saved along with database and behave like a software component. A form provides very easy way (user-friendly) to enter data into database, edit data and display data from database. The non-technical users can also perform various operations on database through forms without going into technical details of a fatabase.

- Report Writers - Most of the DBMSs provide the report writer tools used to create reports. The users can create very easily and quickly. Once a report is created, it can be used may times and it can be modified very easily. The created reports are also saved along with database and behave like a software component.

- Control Over Concurrency - In a computer file-based system, if two users are allowed to access data simultaneously, it is possible that they will interfere with each other. For example, if both users attempt to perform update operation on the same record, then one may overwrite the values recorded by the other. Most database management systems have sub-systems to control the concurrency so that transactions are always recorded with accuracy.

- Backup and Recovery Procedures - In a computer file-based system, the user creates the backup of data regularly to protect the valuable data from damage due to failures to the computer system or application program. It is very time consuming method, if amount of data is large. Most of the DBMSs provide the 'backup and recovery' sub-systems that automatically create the backup of data and restore data if required.

- Data Independence - The separation of data structure of database from the application program that uses the data is called data independence. In DBMS, you can easily change the structure of database without modifying the application program.

## Meaning of "database system

A database system is a high-level definition of the structure and relationship between stored data, a database or databases, users and the hardware or operating system used for the storage

## 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
- Database Access Language

**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

1. DBMS software itself, is the most important software component in the overall system
2. Operating system including network software being used in network, to share the data of database among multiple users.
3. Application programs developed in programming languages such as C++, Visual Basic that are used to to access database in database management system. Each program contains statements that request the DBMS to perform operation on database. The operations may include retrieving, updating, deleting data etc . The application program may be conventional or online workstations or terminals.

**Hardware**

Hardware 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.

**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 (or operational) data and the metadata (data about data or description about data).

**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 hot use or run the database management system. These may include.

1. Procedure to install the new DBMS.
2. To log on to the DBMS.
3. To use the DBMS or application program.
4. To make backup copies of database.
5. To change the structure of database.

6. To generate the reports of data retrieved from database.

## Database Access Language

The database access language is used to access the data to and from the database. The users use the database access language to enter new data, change the existing data in database and to retrieve required data from databases. The user write a set of appropriate commands in a database access language and submits these to the DBMS. The DBMS translates the user commands and sends it to a specific part of the DBMS called the Database Jet Engine. The database engine generates a set of results according to the commands submitted by user, converts these into a user readable form called an Inquiry Report and then displays them on the screen. The administrators may also use the database access language to create and maintain the databases.

The most popular database access language is SQL (Structured Query Language). Relational databases are required to have a database query language.

## Users (role of key players in database design and development)

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

1. Application Programmers
2. Database Administrators
3. End-Users

**Application Programmers**

The people who write application programs in programming languages (such as Visual Basic, Java, or C++) to interact with databases are called Application Programmer.

**Database Administrators**

A person who is responsible for managing the overall database management system is called database administrator or simply DBA.

A database developer is an IT professional responsible for working on database technologies. Where database administrators are more focused on routine maintenance and support for an existing database setup, database developers tend to focus more on improving databases, expanding their range or functionality, or otherwise developing submissions for a company's IT architecture.

**End-Users**

The end-users are the people who interact with database management system to perform different operations on database such as retrieving, updating, inserting, deleting data etc.

## Evolution of DBMS

### 1. The sixties and seventies: centralized

DBMS to the sixties and seventies (IBM IMS, IDS Bull, Univac DMS, etc.) Were totally centralized systems, as befits those years operating systems, and *hardware* for which they were made: a large enterprise-wide computer and a network of dumb terminals and memory.

The first DBMS in the sixties, yet they were called and were aimed at facilitating the use of large data sets in which the interrelationships are complex.The archetype of implementation was the *Bill of materials or explosion Parts,* typical in the automotive, construction of spacecraft and related fields. These systems worked only in batches *(batch).*

Appearing keypad terminals, connected to the central computer via a telephone line, they start to build great applications *on-line* transaction processing (OLTP). The DBMS *software* were closely linked to communications and transaction management.

Although to write application programs using high level languages such as Cobol or PL / I, were also available instructions and subroutines specialized to treat BD requiring that the programmer knew many details of physical design, and that made the was very complex programming.

Since the programs were related to the physical level, it should change continuously when changes were made in the design and organization of the database.The basic concern was to maximize performance: response time and transactions per second.

### 2. The Eighties: relational DBMS

Computers *minis,* first, and then *micro computers,* computer spread to virtually all businesses and institutions.

This required the development of applications would be easier.The DBMS of the seventies were too complex and inflexible, and could only use highly qualified personnel.

The emergence of relational DBMS * marks a significant step to facilitate the programming of applications with BD and to ensure that programs are independent of the physical aspects of the database.

* Oracle appears in 1980.

All these factors make greater use of the DBMS.Standardization, in 1986, the SQL language was a veritable explosion of relational DBMS.

Personal computers

During the eighties appear and spread very quickly on personal computers. It also appears these teams single-user *software* (eg, dBase and its derivatives, Access), with which it is very easy to create and use data sets, and that *personal data* are called *bases.* Notice that the fact referred to these early systems DBMS PC is a bit forced, as it does not accept complex structures or

relationships, or could be used in a network that simultaneously serve many different users.But some, over time, have been turning into real DBMS.

### 3. The nineties: distribution, C / S and 4GL

At the end of the eighties, and relational DBMS is used in virtually all businesses. Nevertheless, until the mid-nineties, when needed a high performance have continued to use the DBMS prerelational.

In the late eighties and early nineties, companies have found that their departments have been buying departmental and personal computers, and applications have been making BD. The result has been that within the company there are numerous DBMS BD and several different types or suppliers.This phenomenon of multiplication of the BD and the DBMS has been increased by the fever of mergers.

The need to have an overview of the company and of linking different applications using different BD, together with the ease that give the networks for communication between computers, has led to the current DBMS, which allow a program to work with different BD as if it were a single. This is what is known as distributed database.

This ideal distribution is achieved when the various BD are supported by one brand of DBMS, ie when there is homogeneity.However, this is not so simple if the DBMS are heterogeneous. Today, thanks largely to the standardization of SQL language, DBMS of different brands can be serviced at each other and work together to provide service to an application program. However, in general, in cases of heterogeneity can not be reached to give the program that uses the appearance that it is a single BD.

In addition to this distribution "imposed", wanting to separate the integrated treatment of pre-existing BD, you can also make a distribution "desired" BD designing a physically distributed and replicated with parts in different systems.The basic reasons for which are interested in this distribution are:

**1)** Availability. The availability of a distributed system with a BD can be higher, because if it goes down one of the systems, others still work. If the data residing in the system is not available are replicated on another system, continue to be available. Otherwise, only available data from other systems.

**2)** Cost. A BD can reduce the cost distributed. In the case of a centralized system, all users computers that can be distributed to different and distant geographical areas are connected to the central system via communication lines.The total cost of communications can be reduced by a user to close the data used most often, eg on a computer in your office or even on your personal computer.

The technology is commonly used to distribute data is known as environment (or architecture) Client / Server (C / S). All of the relational DBMS market have been adapted to this environment.

The idea of C / S is simple. Two different processes running on one system or separate systems, they act so that one has the role of client or a service requester, and the other server or service provider.

For example, a program that a user application running on your PC (which is connected to a network) requests some data from a DB that resides on a UNIX computer which, in turn, runs the relational DBMS that manages it.The application program is the client and the DBMS is the server. A client process can request services to multiple servers. A server can receive requests from many customers. In general, a process that makes customer requesting a service to another process B can also do a service server which prompted another process C (or B, that this request would be the client).Even the client and server can reside on one system.

The ease of distribution of data available is not the only reason, not even the basic, the great success of the environments C / S in the nineties. Perhaps the main reason was the flexibility to build and grow the global computer configuration of the company, as well as making modifications to it, using very standard *hardware* and *software* and cheap.

The success of BD, including personal computers, has led to the emergence of the *Fourth Generation Languages* (4GL), very easy and powerful languages, specializing in application development based on BD.They provide many facilities at the time to define, usually visually, talks to enter, modify, and query data to the C / S.

## Current Trends

Today, relational DBMS are undergoing transformation to accommodate three recent successful technologies, closely related: multimedia, object-oriented (OO) and Internet and web.

The types of data that can be defined in relational DBMS of the eighties and nineties are very limited. The incorporation of multimedia technologies, image and sound-in makes it necessary to accept relational DBMS attributes of these types.

However, some applications do not have enough with the addition of specialized media types.Need complex types that the developer can define as the application. In short, we need to abstract data types: TAD. The latest DBMS already incorporated this possibility, and a wide open market or TAD predefined class libraries.

This brings us to the object-oriented (OO). The success of the OO at the end of the eighties, the development of basic software applications in industrial engineering and construction of graphical interfaces with users, has made during the nineties is widespread in virtually all the fields of computing.

In the SI is also initiated the adoption, shy of the moment, the OO.The use of languages like C + + or Java requires relational DBMS fit them with appropriate interfaces.

The rapid adoption of the SI web makes the DBMS server resources to be incorporated into websites, such as SQL scripts including HTML, Java embedded SQL, etc.. Notice that in the world of the web are common OO data and multimedia.

In recent years it has begun rolling out an application type of the BD called Data Warehouse, or data warehouse, which also produces some changes in the relational DBMS market.

Over the years I have worked with BD in different applications, companies have accumulated large amounts of data of all kinds.If these data are analyzed appropriately can provide valuable information *.

Therefore, it is a great BD keeping with information from all kinds of enterprise applications (and even outside). The data in this big warehouse, Data Warehouse, you get a more or less elaborate replication of which is in the BD used in the daily work of the company. These data warehouses are used exclusivelymind to make inquiries, most especially to carry out studies * financial analysts, market analysts, etc..

Currently, the DBMS is adapted to this type of application, including, for example, tools such as:

a) The creation and maintenance of aftershocks, with some data processing.

b) Consolidation of data from different sources.

c) The creation of physical structures that efficiently support multidimensional analysis.

* For example, market developments in relation to pricing policy.
* These are often multi-dimensional statistics.

# Note: BD refer to Big Data

## Definition of the database approach

The database approach is a way in which data is stored within a computer. It is organized into various charts that are accessed by a variety of computer applications from different locations. Databases are composed of a variety of information that is pertinent and relevant to the organization that is using the database

Database Approach vs. Traditional File Processing

- Self contained nature of database systems (database contains both data and meta-data).
- Data Independence: application programs and queries are independent of how data is actually stored.
- Data sharing.
- Controlling redundancies and inconsistencies.

- Secure access to database; Restricting unauthorized access.
- Enforcing Integrity Constraints.
- Backup and Recovery from system crashes.
- Support for multiple-users and concurrent access.

# CHAPTER 2: DATABASE ORGANISATION

## Meaning of Database Organization

Definition of a database organization refers to the format of the user attribute tables as well as the specification of any topological structures such as layers and networks.

## Database Organization – Types/Approaches

Four main types of database organization:

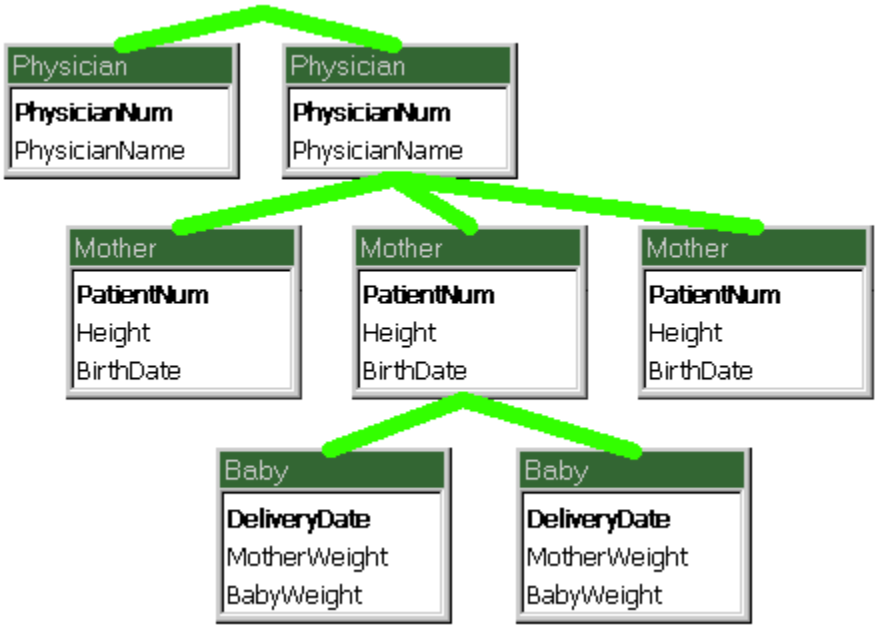
- Flat
- Hierarchical
- Relational
- Object-oriented

### Flat databases

A single kind of record with a fixed number of fields.

Notice the repetition of data, and thus an increased chance of errors.

### Hierarchical databases




Hierarchical relationships among different types of data.

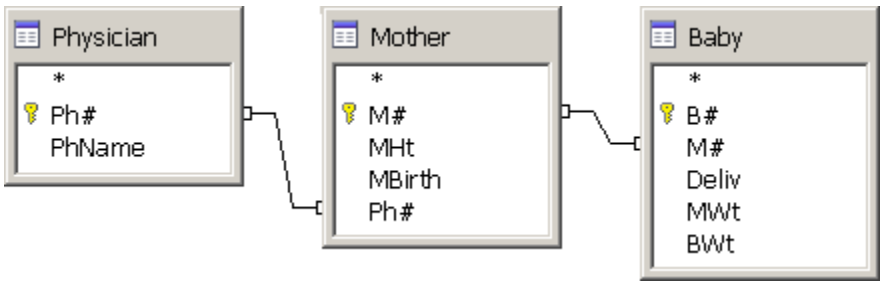
- can be very easy to answer some questions, but very difficult to answer others

- if one-to-many relationship is violated (e.g., a patient can have more than one physician) then the hierarchy becomes a network

### Relational databases

Data are organized as logically independent tables. Features:

- 'Natural'
- Not so strongly biased towards specific questions
- Expresses relationships by means of shared data rather than explicit pointers
- Theoretical basis: relational algebra, calculus; closure
- Operations on tables (Join, Project, Select) to form new tables




### Object-oriented databases

*Object-oriented analysis* is another way to model the world,
involving *abstraction*, *encapsulation*, *modularity* and *hierarchy* (with *inheritance*).

An *object* consists of data and *methods*.

*Classes* are used to group objects which have the same types of data and the same methods.

# CHAPTER 3: PRINCIPLES AND TECHNIQUES OF DATABASE DESIGN

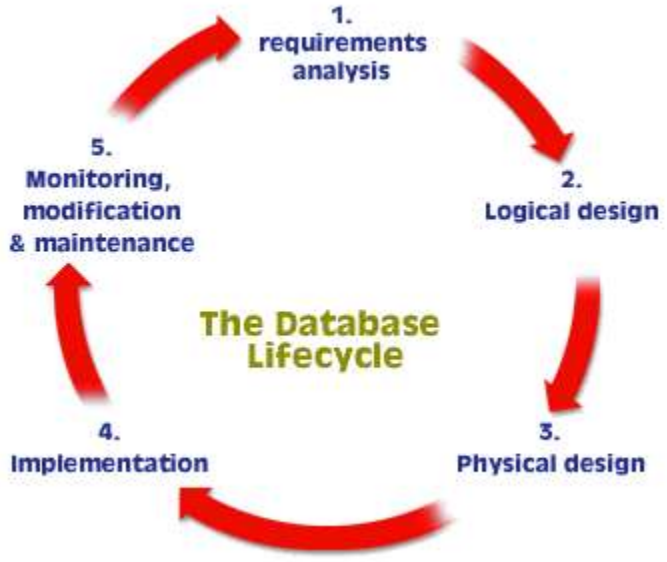
## Introduction to Database design Principles

Is the process of producing a detailed data model of a database. This logical 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.

## The database life cycle (DBLC)

The database life cycle (DBLC) defines the stages involved in getting any type of database off the drawing board and up and running.
In fact, the DBLC never ends because database monitoring, modification, and maintenance are part of the life cycle, and these activities continue long after a database has been implemented.
Put simply, the DBLC encompasses the lifetime of the database.
The five stages in the database life cycle are:

1. Requirements analysis
2. Logical design
3. Physical design
4. Implementation
5. Monitoring, modification, and maintenance




The first three stages are database-design stages, which are briefly described below.

**I. Requirements analysis**

Requirements Analysis is the first and most important stage in the *Database Life Cycle*.
It is the most labor-intensive for the database designer.

This stage involves assessing the informational needs of an organization so that a database can be designed to meet those needs.

**II. Logical design**

During the first part of Logical Design, a *conceptual model* is created based on the needs assessment performed in stage one. A conceptual model is typically an *entity-relationship (ER) diagram* that shows the tables, fields, and primary keys of the database, and how tables are related (linked) to one another.

The tables sketched in the ER diagram are then *normalized*. The *normalization* process resolves any problems associated with the database design, so that data can be accessed quickly and efficiently.

1. *conceptual model:* A description of the structure of a database.
2. *entity-relationship (ER) diagram:* A diagram used during the design phase of database development to illustrate the organization of and relationships between data during database design.
3. *normalization:* The process of applying increasingly stringent rules to a relational database to correct any problems associated with poor design.

**III. Physical design**

The Physical Design stage has only one purpose: to maximize database efficiency.

This means finding ways to speed up the performance of the RDBMS. Manipulating certain database design elements can speed up the two slowest operations in an RDBMS: retrieving data from and writing data to a database.

**IV. Implementation and Monitoring, Modification, and Maintenance**

The final two stages in the DBLC, Implementation and Monitoring, Modification, and Maintenance, occur after the database design is complete.

The following paragraphs discuss these stages in detail.

*Implementation*

During the implementation stage of the DBLC, the tables developed in the ER diagram (and subsequently normalized) are converted into SQL statements and "fed" into the RDBMS to create a database. By this stage in the DBLC, the *System Administrator* has installed and configured an RDBMS.

*System administrator:*

The person responsible for administering a multi-user computer system; duties range from setting up and configuring system components (e.g., an RDBMS) to performing maintenance procedures (e.g., database backups) on the system.

Certain database design books consider converting an ER diagram into SQL statements to be the final task in the logical-design stage. According to such books, implementation is just a matter of feeding SQL statements into an RDBMS and populating the database with data. The difference is not especially important.

*Monitoring, modification, and maintenance*

A successfully implemented database must be carefully *monitored* to ensure that it's functioning properly and that it's secure from unauthorized access. The RDBMS usually provides utilities to help monitor database functionality and security.

Database *modification* involves adding and deleting records, importing data from other systems (as needed), and creating additional tables, user views, and other objects and tools. As an organization grows, its *information system* must grow to remain useful.

information system: Interrelated components (e.g., people, hardware, software, databases, telecommunications, policies, and procedures) that input, process, output, and store data to provide an organization with useful information. Well-designed Database

A well-designed database enhances the organization's ability to expand its information system.

Ongoing *maintenance* procedures include periodic database backups, for example, an important and ongoing maintenance procedure. Again, the RDBMS provides utilities to assist in this task.

## Basic Design Principles

### Avoid redundant information

I.e., keep duplication of data to a minimum. The process to find such a design is called normalization. The normalization process can be formalized to a set of strict rules, and the level of normalization can be expressed in terms of different normal forms, e.g., First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), Boyce-Codd Normal Form (BCNF), Fourth Normal Form (4NF), and Fifth Normal Form (5NF). However, in most cases it is enough to follow the simple rule "avoid redundant information" with your common sence. This is the approach in this material.

### Make the database structure flexible to meet the future needs

Flexibility here means scalability and such design that new tables or table columns need not to be created after the database has been set up. The aim is to be able to handle all situations just by manipulating data, making new data definitions (creation of new tables or fields) unnecessary.

### Define data types and constraints

The three basic data type categories are string, numeric, and date/time*. Choose numeric for fields including values that are measures. For example, phone numbers and zip codes are not a measures, and the appropriate data type is thus string. Date and time data is not string data, even though it is most often represented as strings to the user. After the data type cathegory has been chosen, next task is to find the appropriate (maximum) length for the data items, for numeric and date/time data types also the accuracy.

The main conciderations with respect to defining constraints:

- is the field value always required, i.e., are NULL values denied (NOT NULL constraint in SQL)**
- should the field values be unique (PRIMARY KEY constraint, UNIQUE constraint)***
- what set of values is allowed (CHECK constraint, FOREIGN KEY constraint)
- is there some more complex constraints or business rules to be set up (ASSERTION constraint, triggers)

* Techically speaking, date and time data types fall into the cathegory of numeric data types.
** Make sure that you understand the difference between empty string and NULL. With string data types they can look the same from the user's point of view, but they behave differently with search conditions. To avoid confusion, some developers advice to deny NULLs in fields with string data type.
*** Table can have only one primary key but many alternate keys, i.e., keys defined with the UNIQUE constraint. Notice that both constraints can involve a set of fields rather than only one field. Make sure that you understand the difference between these two:
a) table has two UNIQUE constraints: UNIQUE(a), UNIQUE(b);
b) table has one UNIQUE constraint: UNIQUE(a,b).

**Optimize physical performance**

Speed is a very improtant factor in database applications. Indexes help the database management system (DBMS) to search data from the database tables. Indexes are data structures maintained automatically by the DBMS after they have been created. Primary indexes, indexes for the primary key fields, are automatically created. Normally, other indexes have to be created manually. It is not always easy to analyse what indexes are worth creating. Namely, even though indexes speed up search queries*, they slow down update operations (because each update means that also the index has to be updated).
* However, not in all cases. For example, if all rows or relatively very large number of rows in a table are selected to the result set of the query, it is obviously more efficient to search the file from the beginning to the end sequentially without using any indexes.

Another way to speed up data searches is denormalization, in the case that table joins should be made in the query. Denormalization simply means joining two or more tables into one table. Then the join operation hasn't have to be done by the query, which saves time. Again, this means slowing down data update operations, because in denormalized tables there are redundancies: same information has to be updated in many places.

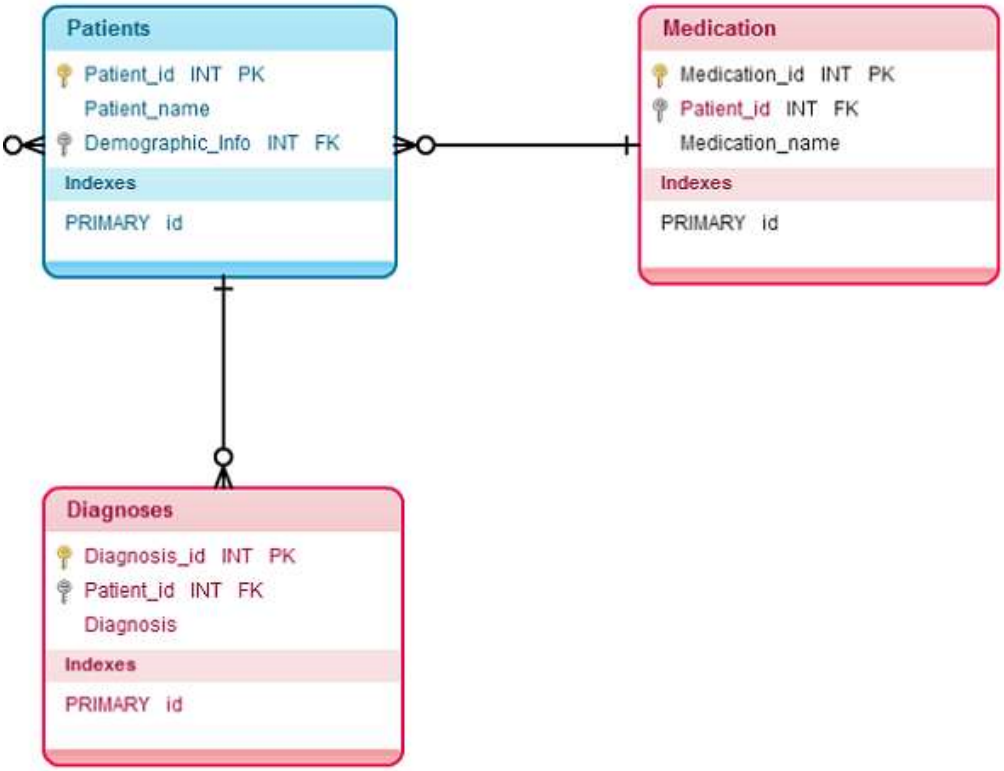
This material concentrates on designing operational databases, databases where update operations take place often, and it is important that updates are made fluently. Thus, denormalization is not normally a good option. If we were designing data warehouse databases, denormalization would be a preferred operation. In data warehouses, data searches to a huge amount of history data have to be carried out as fast as possible, but data updates are batch processed, e.g., at nights, and thus can be more slow.

## Types of Database Modeling Techniques

Below is a list of the most common database modeling methods. Do note that, depending on the type of data and end user needs when accessing the database, it’s possible to employ multiple models to create a more sophisticated database design. Of course, in either scenario, the production of database diagrams would be required to establish and maintain high operational standards.

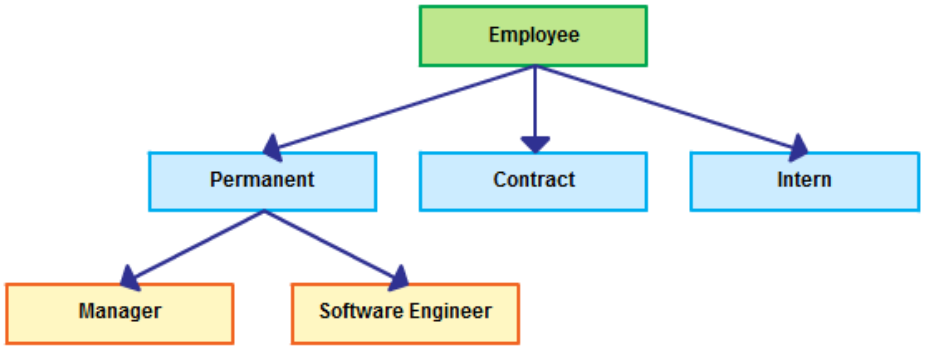
From the below mentioned models the relational model is the most commonly used model for most database designs. But in some special cases other models can be more beneficial.

- **Relational Model**: Founded on mathematical theory, this database model takes information storage and retrieval to a new level because it offers a way to find and understand different relationships between the data. By looking at how different variables can change the relationship between the data, new perspectives can be gained as the information's presentation is altered by focusing on different attributes or domains. These models can often be found within airline reservation systems or bank databases.



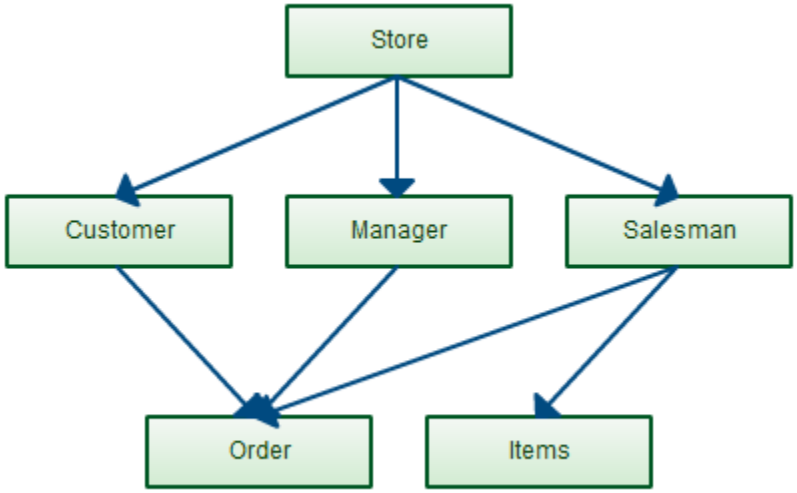

Relational design method, the most popular database design method

- **Graph Model**: Graph model is another model that is gaining popularity. These databases are created based on the Graph theory and used nodes and edges to represent data. The structure is somewhat similar to object oriented applications. Graph databases are generally easier to scale and usually perform faster for associative data sets.
- **Hierarchical Model**: Much like the common organizational chart used to organize companies, this database model has the same tree-like appearance and is often used to structure XML documents. In looking at data efficiency, this is an ideal model where the data contains nested and sorted information, but it can be inefficient when the data does not have an upward link to a main data point or subject. This model works well for an employee information management system in a company that seeks to restrict or assign equipment usage to certain individuals and/or departments.




Hierarchical method, the very first database design model

- **Network Model**: Using records and sets, this model uses a one-to-many relationship approach for the data records. Multiple branches are allocated for lower-level structures and branches that are then connected by multiple nodes, which represent higher-level structures within the information. This database modeling method provides an efficient way to retrieve information and organize the data so that it can be looked at multiple ways, providing a means of increasing business performance and reaction time. This is a viable model for planning road, train, or utility networks.




The network model where a node can have multiple parent nodes

- **Dimensional Model**: This is an adaptation of the relational model and is often used in conjunction with it by adding the “dimension” of fact to the data points. Those facts can be used as measuring sticks for the other data to determine how a size of a group or the timing of a group impacted upon certain results. This can help a business make more effective strategic decisions and help them get to know their target audience. These models can be useful to organizations with sales and profit analysis.

- **Object Relational Model**: These models have created an entirely new type of database, which combines database design with application program to solve specific technical problems while leveraging the best of both worlds. To date, object databases still need to be refined to achieve greater standardization. Real world applications of this model often include technical or scientific fields, such as engineering and molecular biology.

### Database Diagrams and Model Selection

No matter which database modeling method you choose, it's imperative to develop related diagrams to visualize the desired flow and functionality to ensure the database is designed in the most efficient and effective way possible. The right diagram will reduce revisions and rework because you can test the proposed design before putting in the time and expense of actually creating it. Diagrams are also a highly effective communication tool, particular for large teams, as they facilitate clear and quick communication.

# CHAPTER 4: RELATIONAL DATABASE SYSTEM

## Introduction to relational database management system

**RDBMS (relational database management system),** A database based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organised in tables.

**A relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables**

## Characteristics of Relational Databases

CODD'S RULES FOR RELATIONAL DATABASES, The relational model for databases described by Dr. Codd contains 12 rules.

### Relational Database Characteristics

1) Data in the relational database must be represented in tables, with values in columns within rows.
2) Data within a column must be accessible by specifying the table name, the column name, and the value of the primary key of the row.
3) The DBMS must support missing and inapplicable information in a systematic way, distinct from regular values and independent of data type.
4) The DBMS must support an active on-line catalogue.
5) The DBMS must support at least one language that can be used independently and from within programs, and supports data definition operations, data manipulation, constraints, and transaction management.
6) Views must be updatable by the system.
7) The DBMS must support insert, update, and delete operations on sets.
8) The DBMS must support logical data independence.
9) The DBMS must support physical data independence.
10) Integrity constraints must be stored within the catalogue, separate from the application.
11) The DBMS must support distribution independence. The existing application should run when the existing data is redistributed or when the DBMS is redistributed.
12) If the DBMS provides a low level interface (row at a time), that interface cannot bypass the integrity constraints.

## Relational algebra

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

Fundamental operations of Relational algebra:

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

These are defined briefly as follows:

**Select Operation (σ)**

Selects tuples that satisfy the given predicate from a relation.

**Notation σ*p*(r)**

Where *p* stands for selection predicate and r stands for relation. *p* is prepositional logic formulae which may use connectors like and, or and not. These terms may use relational operators like: =, ≠, ≥, < , >, ≤.

For example:

σ*subject="database"*(**Books**)

Output : Selects tuples from books where subject is 'database'.

σsubject="database" and price="450"(Books)

Output: Selects tuples from books where subject is 'database' and 'price' is 450.

σsubject="database" and price < "450" or year > "2010"(Books)

Output : Selects tuples from books where subject is 'database' and 'price' is 450 or the publication year is greater than 2010, that is published after 2010.

**Project Operation (Π)**

Projects column(s) that satisfy given predicate.

**Notation: ΠA1, A2, An (r)**

Where a1, a2 , an are attribute names of relation r.

Duplicate rows are automatically eliminated, as relation is a set.

for example:

$\Pi_{subject,\ author}(Books)$

Selects and projects columns named as subject and author from relation Books.

## Union Operation (∪)

Union operation performs binary union between two given relations and is defined as:

$$r \cup s = \{ t \mid t \in r \text{ or } t \in s\}$$

**Notion: r U s**

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

- r, s must have same number of attributes.
- Attribute domains must be compatible.

Duplicate tuples are automatically eliminated.

$$\Pi \text{ author (Books)} \cup \Pi \text{ author (Articles)}$$

Output: Projects the name of author who has either written a book or an article or both.

## Set Difference ( − )

The result of set difference operation is tuples which present in one relation but are not in the second relation.

**Notation: r − s**

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

$$\Pi \text{ author (Books)} - \Pi \text{ author (Articles)}$$

Output: Results the name of authors who has written books but not articles.

## Cartesian Product (X)

Combines information of two different relations into one.

**Notation: r X s**

Where r and s are relations and there output will be defined as:

$$r \ X \ s = \{ q \ t \mid q \in r \text{ and } t \in s\} \ \Pi_{\text{ author = 'tutorialspoint'}}(Books \ X \ Articles)$$

Output : yields a relation as result which shows all books and articles written by tutorialspoint.

## Rename operation ( ρ )

Results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. rename operation is denoted with small greek letter rho *ρ*
**Notation: *ρ* x (E)**
Where the result of expression E is saved with name of x.
Additional operations are:

- Set intersection
- Assignment

Natural join

## Relational Algebra Types and Operations

An algebra is a formal structure consisting of *sets* and *operations* on those sets.

Relational algebra is a formal system for manipulating relations.

- Operands of this algebra are relations.
- Operations of this algebra include the usual set operations (since relations are sets of tuples), and special operations defined for relations
    - selection
    - projection
    - join

Relational algebra is a formal system for manipulating relations (Tables). Operands of this algebra are relations. Operations of this algebra include the usual set operations (since relations are sets of tuples [Rows]), and special operations defined for relations.

---

### Set Operations on Relations

For the set operations on relations, both operands must have the same scheme, and the result has that same scheme.

- R1 U R2 (union) is the relation containing all tuples that appear in R1, R2, or both.
- R1 n R2 (intersection) is the relation containing all tuples that appear in both R1 and R2.
- R1 - R2 (set difference) is the relation containing all tuples of R1 that do not appear in R2.

---

### Selection $(\sigma)$

Selects tuples from a relation whose attributes meet the selection criteria, which is normally expressed as a predicate.

```
R2 = select(R1,P)
```

That is, from R1 we create a new relation R2 containing those tuples from R1 that satisfy (make true) the predicate P.

A *predicate* is a boolean expression whose operators are the logical connectives (and, or, not) and arithmetic comparisons (LT, LE, GT, GE, EQ, NE), and whose operands are either domain names or domain constants.

select(Workstation,Room=633) =

```
Name     Room     Mem    Proc   Monitor
==================================
coke      633   16384   SP4    color17
bass      633    8124   SP2    color19
bashful   633    8124   SP1    b/w
```

select(User,Status=UG and Idle<1:00) =

```
Login    Name         Status    Idle   Shell   Sever
=============================================
jli      J. Inka      UG        0:00   bsh     UG
```

---

## Projection (Π)

Chooses a subset of the columns in a relation, and discards the rest.

```
R2 = project(R1,D1,D2,...Dn)
```

That is, from the tuples in R1 we create a new relation R2 containing only the domains D1,D2,..Dn.

project(Server,Name,Status) =

```
Name     Status
==============
diamond   up
emerald   up
graphite  down
ruby      up
frito     up
```

```
project(select(User,Status=UG),Name,Status) =

Name         Status
==================
A. Cohn        UG
J. Inka        UG
R. Kemp        UG
```

---

Join

Combines attributes of two relations into one.

```
R3 = join(R1,D1,R2,D2)
```

Given a domain from each relation, *join* considers all possible pairs of tuples from the two relations, and if their values for the chosen domains are equal, it adds a tuple to the result containing all the attributes of both tuples (discarding the duplicate domain D2).

Natural join: If the two relations being joined have exactly one attribute (domain) name in common, then we assume that the single attribute in common is the one being compared to see if a new tuple will be inserted in the result.

Assuming that we've augmented the domain names in our lab database so that we use MachineName, PrinterName, ServerName, and UserName in place of the generic domain "Name", then

```
join(Workstations,Printers)
```

is a natural join, on the shared attribute name Room. The result is a relation of all workstation/printer attribute pairs that are in the same room.

---

**Example Use of Project and Join**

Find all workstations in a room with a printer.

- R1 = project(Workstation,Name,Room)
- R2 = project(Printer,Name,Room)
- R3 = join(R1,R2)

```
R1              R2              R3
Name    Room    Name    Room    WName   Pname   Room
==========      ==========      ====================
coke     633    chaucer  737    coke    uglab   633
bass     633    keats    706    bass    uglab   633
bashful  633    poe      707    bashful uglab   633
tab      628    dali     737
crush    628    uglab    633
```

---

## Implementing Set Operations

To implement R1 U R2 (while eliminating duplicates) we can

- sort R1 in O(N log N)
- sort R2 in O(M log M)
- merge R1 and R2 in O(N+M)

If we allow duplicates in union (and remove them later) we can

- copy R1 to R3 in O(N)
- insert R2 in R3 in O(M)

If we have an index and don't want duplicates we can

- copy R1 to R3 in O(N)
- for each tuple in R2 (which is O(M))
    - use index to lookup tuples in R1 with the same index value O(1)
    - if R2 tuple equals some such R1 tuple, don't add R2 tuple to R3

Intersection and set difference have corresponding implementations.

---

### Implementing Projection

To implement projection we must

- process every tuple in the relation
- remove any duplicates that result

To avoid duplicates we can

- sort the result and remove consecutive tuples that are equal
    - requires time O(N log N) where N is the size of the original relation
- implement the result as a set

- set insertion guarantees no duplicates
- by using a hash table, insertion is O(1), so projection is O(N)

---

**Implementing Selection**

In the absence of an index we

- apply the predicate to every tuple in the relation
- insert matches in the resulting relation
  - duplicates can't occur
- take O(N) time

Given an index, and a predicate that uses the index key, we

- Lookup tuples using the key
- evaluate only those tuples with the predicate
- take O(K) time, where K tuples match the key

---

**Implementing Join with Nested Loops**

A *nested loop join* on relations R1 (with N domains) and R2 (with M domains), considers all |R1| x |R2| pairs of tuples.

```
R3= join(R1,Ai,R2,Bj)

for each tuple t in R1 do
  for each tuple s in R2 do
    if t.Ai = s.Bj then
      insert(R3, t.A1, t.A2, ..., t.AN,
        s.B1, ..., s.B(j-1), s.B(j+1), ..., s.BM)
```

This implementation takes time O(|R1|*|R2|).

---

**Index Join**

An *index join* exploits the existence of an index for one of the domains used in the join to find matching tuples more quickly.

```
R3= join(R1,Ai,R2,Bj)

for each tuple t in R1 do
```

```
for each tuple s in R2 at index(t.Ai) do
   insert(R3, t.A1, t.A2, ..., t.AN,
     s.B1, ..., s.B(j-1), s.B(j+1), ..., s.BM)
```

We could choose to use an index for R2, and reverse the order of the loops.

The decision on which index to use depends on the number of tuples in each relation.

---

**Sort Join**

If we don't have an index for a domain in the join, we can still improve on the nested-loop join using *sort join*.

```
R3= join(R1,Ai,R2,Bj)
```

- Merge the tuples of both relations into a single list
  - list elements must identify the original relation
- Sort the list based on the value in the join domains Ai and Bj
  - all tuples on the sorted list with a common value for the join domains are consecutive
- Pair all (consecutive) tuples from the two relations with the same value in the join domains

---

**Comparison of Join Implementations**

Assumptions

- Join R1 and R2 (on domain D) producing R3
- R1 has i tuples, R2 has j tuples
- |R3| = m, 0 <= m <= i * j
- Every implementation takes at least time O(m)

Comparison

- Nested-loop join takes time O(i * j)
- Index join (using R2 index) takes time O(i+m)
  - lookup is O(1) for each tuple in R1
  - at most O(m) tuples match
- Sort join takes time O(m +(i+j)log(i+j))
  - O(i+j) to merge the tuples in R1 and R2
  - O((i+j) log (i+j)) to sort the list
  - O(m) to produce the output (0 <= m <= i*j)

## Expressing Queries in Relational Algebra

Relational algebra is an unambiguous notation (or formalism) for expressing queries.

Queries are simply expressions in relational algebra.

Expressions can be manipulated symbolically to produce simpler expressions according to the laws of relational algebra.

Expression simplification is an important query optimization technique, which can affect the running time of queries by an order of magnitude or more.

- early "selection" reduces the number of tuples
- early "projection" reduces the number of domains

---

**Algebraic Laws for Join**

Commutativity (assuming order of columns doesn't matter)

join(R1, Ai, R2, Bj) = join(R2, Bj, R1, Ai)

Nonassociativity

join (join(R1, Ai, R2, Bj),Bj,R3,Ck)

is not the same as

join (R1,Ai,join(R2, Bj, R3, Ck),Bj)

---

**Algebraic Laws for Selection**

Commutativity

select(select(R1,P1),P2) = select(select(R1,P2),P1)

Selection pushing

- if P contains attributes of R

  select(join(R,Ai,S,Bj),P) = join(select(R,P),Ai,S,Bj)

- if P contains attributes of S

select(join(R,Ai,S,Bj),P) = join(R,Ai,select(S,P),Bj)

Selection Splitting (where P = A and B)

select(R,P) = select(select(R,A),B)

select(R,P) = select(select(R,B),A)

---

**Example: Selection Pushing and Splitting**

Consider the following 4 relation database

- CSG: Course-StudentID-Grade
- SNAP: StudentID-Name-Address-Phone
- CDH: Course-Day-Hour
- CR: Course-Room

Implement the query "Where is Amy at Noon on Monday?"

Let P be (Name="Amy" and Day="Monday" and Hour="Noon")

We can use a brute-force approach that *joins* all the data in the relations into a single large relation, *selects* those tuples that meet the query criteria, and then isolates the answer field using *projection*.

- R1 = join(CSG,SNAP)
- R2 = join(R1,CDH)
- R3 = join(R2,CR)
- R4 = select(R3,P)
- R5 = project(R4,Room)

```
project(select(join(join(join(CSG,SNAP),CDH),CR),P),Room)
```

The selection uses only Name, Day, and Hour attributes (and not Course or Room), so we can push the selection inside the outermost join.

- R1 = join(CSG,SNAP)
- R2 = join(R1,CDH)
- R3 = select(R2,P)
- R4 = join(R3,CR)
- R5 = project(R4,Room)

We cannot push selection further, because the predicate involves attributes from both operands of the next innermost join (R1,CDH).

We can split the selection into two, one based on Name, and the other based on Day-Hour.

- R1 = join(CSG,SNAP)
- R2 = join(R1,CDH)
- R3 = select(R2,Day="Monday" and Hour="Noon")
- R4 = select(R3,Name="Amy")
- R5 = join(R4,CR)
- R6 = project(R5,Room)

Now we can push the first selection inside the join, since it involves only attributes from the CDH relation.

- R1 = join(CSG,SNAP)
- R2 = select(CDH,Day="Monday" and Hour="Noon")
- R3 = join(R1,R2)
- R4 = select(R3,Name="Amy")
- R5 = join(R4,CR)
- R6 = project(R5,Room)

Similarly we can push the second selection inside the preceding join, since it involves only attributes from R1 (ie, Name).

- R1 = join(CSG,SNAP)
- R2 = select(CDH,Day="Monday" and Hour="Noon")
- R3 = select(R1,Name="Amy")
- R4 = join(R2,R3)
- R5 = join(R4,CR)
- R6 = project(R5,Room)

Continuing to push the second select inside the first join

- R1 = select(SNAP,Name="Amy")
- R2 = join(CSG,R1)
- R3 = select(CDH,Day="Monday" and Hour="Noon")
- R4 = join(R2,R3)
- R5 = join(R4,CR)
- R6 = project(R5,Room)

**Algebraic Laws for Projection**

**Projection pushing**

To push a projection operation inside a join requires that the result of the projection contain the attributes used in the join.

project(join(R,Ai,S,Bj),D1,D2,...Dn)

In this case, we know that the domains in the projection will exist in the relation that results from the join.

In performing projection first (on the two join relations)

- we should only project on those domains that exist in each of the two relations
- we must ensure that the join domains Ai and Bj exist in the resulting two relations

Let PDR = {D|D domain in R, D in {D1...Dn}} U Ai

Let PDS = {D|D domain in S, D in {D1...Dn}} U Bi

R1 = project(R,PDR)

R2 = project(S,PDS)

R3 = join(R1,Ai,R2,Bj) = project(join(R,Ai,S,Bj),D1,D2,...Dn)

---

**Example: Projection Pushing**

Implement the query "Where is Amy at Noon on Monday?"

- R1 = select(SNAP,Name="Amy")
- R2 = join(CSG,R1)
- R3 = select(CDH,Day="Monday" and Hour="Noon")
- R4 = join(R2,R3)
- R5 = join(R4,CR)
- R6 = project(R5,Room)

This approach carries along unnecessary attributes every step of the way.

- R1 carries Address and Phone attributes
- R4 carries Grade attribute

We use projection pushing to eliminate unnecessary attributes early in the implementation.

- R1 = select(SNAP,Name="Amy")
- R2 = join(CSG,R1)
- R3 = select(CDH,Day="Monday" and Hour="Noon")
- R4 = join(R2,R3)
- R5 = project(CR, Course, Room)
- R6 = project(R4, Course)
- R7 = join(R5,R6)
- R8 = project(R7,Room)

Note that R5 is unnecessary, since the domains in the projection are all the domains of CR.

Implement the query "Where is Amy at Noon on Monday?"

- R1 = select(SNAP,Name="Amy")
- R2 = join(CSG,R1)
- R3 = select(CDH,Day="Monday" and Hour="Noon")
- R4 = join(R2,R3)
- R5 = project(R4, Course)
- R6 = join(CR,R5)
- R7 = project(R6,Room)

We can continue pushing the projection on Course below the join for R4.

- R1 = select(SNAP,Name="Amy")
- R2 = join(CSG,R1)
- R3 = select(CDH,Day="Monday" and Hour="Noon")
- R4 = project(R2,Course)
- R5 = project(R3,Course)
- R6 = join(R4,R5)
- R7 = join(CR,R6)
- R8 = project(R7,Room)

We can continue pushing the projection on Course for R4 below the join for R2.

- R1 = select(SNAP,Name="Amy")
- R2 = project(CSG,Course,StudentID)
- R3 = project(R1,StudentID)
- R4 = join(R2,R3)
- R5 = project(R4,Course)
- R6 = select(CDH,Day="Monday" and Hour="Noon")
- R7 = project(R6,Course)
- R8 = join(R6,R7)
- R9 = join(CR,R8)
- R10 = project(R9,Room)

## Relational calculus

An operational methodology, founded on predicate calculus, dealing with descriptive expressions that are equivalent to the operations of relational algebra. Codd's reduction algorithm can convert from relational calculus to relational algebra.
Two forms of the relational calculus exist: the tuple calculus and the domain calculus.
predicate calculus is the system of symbolic logic concerned not only with relations between propositions as wholes but also with the representation by symbols of individuals and predicates in propositions and with quantification over individuals Also called functional calculus

In contrast with Relational Algebra, Relational Calculus is non-procedural query language, that is, it tells what to do but never explains the way, how to do it.
Relational calculus exists in two forms:

### The Tuple Relational Calculus [TRC]

1. The tuple relational calculus is a nonprocedural language. (The relational algebra was procedural.)

   We must provide a formal description of the information desired.

2. A query in the tuple relational calculus is expressed as

   $$\{t \mid P(t)\}$$

   i.e. the set of tuples $t$ for which predicate $P$ is true.

3. We also use the notation
   - $t[a]$ to indicate the value of tuple $t$ on attribute $a$.
   - $t \in r$ to show that tuple $t$ is in relation $r$.

#### Tuple relational calculus Implementation

Filtering variable ranges over tuples
**Notation: { T | Condition }**
Returns all tuples T that satisfies condition.

For Example:

```
{ T.name | Author(T) AND T.article = 'database' }
```

Output: returns tuples with 'name' from Author who has written article on 'database'.
TRC can be quantified also. We can use Existential ( ∃ )and Universal Quantifiers ( ∀ ).

For example:

```
{ R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}
```

Output : the query will yield the same result as the previous one.

## The Domain Relational Calculus [DRC]

Domain variables take on values from an attribute's domain, rather than values for an entire tuple.

A formal query in DRC is expressed as {<x1, x2 , …, xn> | P(x1, x2 , …, xn )} where x1, x2 , …, xn are domain variables and P(x1, x2 , …, xn) is a formula involving those variables.

Formal definition of DRC formula

Similar to the TRC, formula in Domain Relational Calculus is built up from atoms. An atom in the Domain Relational Calculus has one of the following forms:

- $< x1 , x2 , . . . , xn > \in r$ where r is a relation of n attributes and x1, x2 , …, xn are domain variables or domain constant.

- x op y where x, y are domain variables , op is a comparison operation. Note that x, y have domains that can be compared by op
- x op const where x is a domain variable and const is a constant in domain of attribute for which x is a variable.

The following rules are used to build up the Domain Relational Calculus formula from atoms:

- An atom is a formula
- If P is a formula , then so are ¬P and (P)
- If P1 and P2 are formulae, then so are P1∧P2 , P1∨P2and P1 ⇒ P2
- If P(t) is a formula in x where x is a domain variable then ∃x (P(x)) and ∀x(P(x)) are also formulae.

**Domain relational calculus implementation**
In DRC the filtering variable uses domain of attributes instead of entire tuple values (as done in TRC, mentioned above).
**Notation:**
**{ a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}**

where a1, a2 are attributes and P stands for formulae built by inner attributes.
For example: {< article, page, subject > | ∈ TutorialsPoint ∧ subject = 'database'}
Output: Yields Article, Page and Subject from relation TutorialsPoint where Subject is database.
Just like TRC, DRC also can be written using existential and universal quantifiers. DRC also involves relational operators.
Expression power of Tuple relation calculus and Domain relation calculus is equivalent to Relational Algebra.

Example queries

Query 1: Find all employees whose salary is greater than 30.000

{ < id,n,b,a,s,d > | < id,n,b,a,s,d > ∈ EMPLOYEE ∧ s > 30 . 000 }

Query 2: Find the name, address of employees who works for department number 1

{ < n,a > | ∃ id,b,s,d ( < id,n,b,a,s,d > ∈ EMPLOYEE ∧ d = 1 ) }

Query 3: Find the name of the department that employee John works for.

{ < dn > | ∃ di,o,m ( < di,dn,o,m > ∈ DEPARTMENT ∧ ∃ id,n,b,a,s,d ( < id, n, b, a, s,
d > ∈ EMPLOYEE ∧ ( n = John ∧ d = di ) ) ) }

Query 4: Find the SSN, start date of the employees who works for project number P1 or project number P2

{ < en,sd > | ∃ p ( < en,p,sd > JOIN ∧ ( p = 1 ∨ p = 2 ) ) }

Query 5: Find the name, relationship of all the dependents of employees who works for Department Human Resource

{ en,name, r | ∃ bd ( < en,name,bd,r > ∈ EMP-DEPENDENT ∧

∃ id,n,b,a,s,d ( < id, n, b, a, s,d > ∈ EMPLOYEE ∧

( en = id ∧ ∃ di, dn, o, m ( < di, dn, o, m > ∈ DEPARTMENT ∧ di = d ∧ dn = 'Human Resource' ) ) ) ) }

Query 6: Finds the name of the employees who join in every project.

{ name | ∃ id,n,b,a,s,d ( < id, n, b, a,
s,d > ∈ EMPLOYEE ∧ ∀ c,pn,b,dept ( < c,pn,b,dept > ∈ PROJECT

⇒ ( ∃ emp, proj, s ( < emp, proj, s > ∈ JOIN ∧ proj = c ∧ emp = id ) ) ) ) }

Query 7: Finds the names of the employees who have no dependent

n { | ∃ id,n,b,a,s,d ( < id,n, b, a, s,d > ∈ EMPLOYEE ∧

¬ ( ∃ en, dn, bd, r ( < en,dn,bd,r > ∈ EMP-DEPENDENT ∧ id = en ) ) ) }

## Safety of the expression

As we mentioned before, in Tuple Relational Calculus, it is possible to write expressions that may generate infinite relation. The same situation arises for Domain Relational Calculus.

In the Domain Relational Calculus, we must concerned about the formular within “there exists” and “for all”. Consider the expression {x|∃(<x,y>r)∧∃z(¬(<x,z>r)∧P(x,z))}. In the second part of this expression ∃z(¬(<x,z>r)∧P(x,z)) we need to consider values for z that do not appear in r. This set of values actually an infinite set. This case does not appear in the Tuple relational Calculus because in TRC the universal and existential quantifiers are applied on variables that already range over a specific relation ( in the form ∃t r (P(t)) and ∀t r (P(t)) ) .

Therefore, in DRC, an expression {<x1, x2 , …, xn> | P(x1, x2 , …, xn )} is safe if all the following hold:

- All values in the result are values from DOM(P)
- For every “there exists” sub formula of the form ∃x (P1(x)), the sub-formula is true iff there is a value x in DOM(P1) such that P1(x) is true
- For every “for all” sub formula of the form ∀x(P2(x)), the su-bformula is true iff for all values x from DOM(P2) , P2(x) is true

## Expressive Power of Languages

The Domain Relational Calculus restricted to safe expression, is equivalent to relational algebra in term of expressive power.

# CHAPTER 5: ENTITY RELATIONSHIP

## Introduction and meaning of Entity Relationship

In software engineering, an entity–relationship model (ER model) is a data model for describing the data or information aspects of a business domain or its process requirements, in an abstract way that lends itself to ultimately being implemented in a database such as a relational database.

## Connotations of Entity Relationship

An entity–relationship model is a systematic way of describing and defining a business process. The process is modeled as components (*entities*) that are linked with each other by *relationships* that express the dependencies and requirements between them, such as: *one building may be divided into zero or more apartments, but one apartment can only be located in one building.* Entities may have various properties (*attributes*) that characterize them. Diagrams created to represent these entities, attributes, and relationships graphically are called entity–relationship diagrams.

An ER model is typically implemented as a database. In the case of a relational database, which stores data in tables, every row of each table represents one instance of an entity. Some data fields in these tables point to indexes in other tables; such pointers represent the relationships.

The three schema approach to software engineering uses three levels of ER models that may be developed.

Conceptual data model

This is the highest level ER model in that it contains the least granular detail but establishes the overall scope of what is to be included within the model set. The conceptual ER model normally defines master reference data entities that are commonly used by the organization. Developing an enterprise-wide conceptual ER model is useful to support documenting the data architecture for an organization.

A conceptual ER model may be used as the foundation for one or more *logical data models* (see below). The purpose of the conceptual ER model is then to establish structural metadata commonality for the master data entities between the set of logical ER models. The conceptual data model may be used to form commonality relationships between ER models as a basis for data model integration.

Logical data model

A logical ER model does not require a conceptual ER model, especially if the scope of the logical ER model includes only the development of a distinct information system. The logical ER model contains more detail than the conceptual ER model. In addition to master data entities, operational and transactional data entities are now defined. The details of each data entity are developed and the entity relationships between these data entities are established. The logical ER model is however developed independent of technology into which it is implemented.

Physical data model

One or more physical ER models may be developed from each logical ER model. The physical ER model is normally developed to be instantiated as a database. Therefore, each physical ER

model must contain enough detail to produce a database and each physical ER model is technology dependent since each database management system is somewhat different. The physical model is normally instantiated in the structural metadata of a database management system as relational database objects such as database tables, database indexes such as unique key indexes, and database constraints such as a foreign key constraint or a commonality constraint. The ER model is also normally used to design modifications to the relational database objects and to maintain the structural metadata of the database.

## Data Models

Before you look at specific symbols, it's important to understand the various levels of ERDs. There are several ways to model entity-relationship diagrams. The most high-level type is a conceptual data model; the next highest is the logical data model, and the lowest-level (and therefore most detailed) type is the physical data model. Consult the chart below to see which elements are covered in each data model.

| Feature | Conceptual | Logical | Physical |
|---|---|---|---|
| Entity names | X | X | |
| Entity relationships | X | X | |
| Attributes | | X | |
| Primary keys | | X | X |
| Foreign keys | | X | X |
| Table names | | | X |
| Column names | | | X |
| Column data types | | | X |

### CONCEPTUAL DATA MODEL

This ER model establishes a broad view of what should be included in the model set. Conceptual data models:

- Include important entities and the relationship between them.
- Do not specify attributes.
- Do not specify primary keys.

Conceptual ERDs can be used as the foundation for logical data models. They may also be used to form commonality relationships between ER models as a basis for data model integration.

### LOGICAL DATA MODEL

This model contains more detail than the conceptual ER model, without regard to how information will be physically implemented in the database. Logical data models:

- Include all entities and relationships between them.

- Specify attributes for each entity.
- Specify primary key for each entity.
- Specify foreign keys, which identify the relationship between different entities.
- Involve normalization, which is the process of removing redundancy in a table so that the table is easier to modify. Normalization typically occurs by dividing an entity table into two or more tables and defining relationships between the tables.

### PHYSICAL DATA MODEL

The physical data model represents the process of adding information to the database. This model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables. Physical data models:

- Specify all tables and columns.
- Include foreign keys to identify relationships between tables.
- May include denormalization, depending on user requirements.
- May be significantly different from the logical data model.
- Will differ depending on which DBMS (database management system) is used.

## Drawing ERD

### Conceptual ERD Symbols

These symbols are generally used for conceptual data models, although some aspects may spill over into logical data models. They can be found in the UML Entity Relationship and Entity Relationship shape libary of Lucidchart. If you don't see the shape you need, use an image file (Lucidchart supports .PNG, .JPG, or .SVG import) or create your own with our existing shapes and styling options.

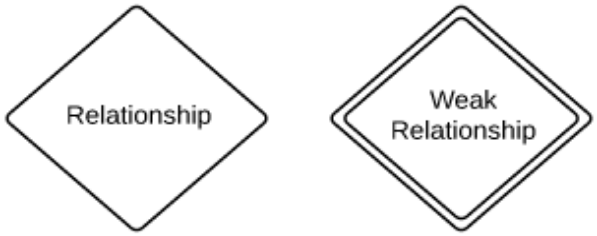
#### ENTITIES

Entities are objects or concepts that represent important data. They are typically nouns, e.g. *customer*, *supervisor*, *location*, or *promotion*.

- Strong entities exist independently from other entity types. They always possess one or more attributes that uniquely distinguish each occurrence of the entity.
- Weak entities depend on some other entity type. They don't possess unique attributes (also known as a primary key) and have no meaning in the diagram without depending on another entity. This other entity is known as the owner.
- Associative entities are entities that associate the instances of one or more entity types. They also contain attributes that are unique to the relationship between those entity instances.

**RELATIONSHIPS**

- Relationships are meaningful associations between or among entities. They are usually verbs, e.g. *assign*, *associate*, or *track*. A relationship provides useful information that could not be discerned with just the entity types.
- Weak relationships, or identifying relationships, are connections that exist between a weak entity type and its owner.




**ATTRIBUTES**

- Attributes are characteristics of either an entity, a many-to-many relationship, or a one-to-one relationship.
- Multivalued attributes are those that are capable of taking on more than one value.
- Derived attributes are attributes whose value can be calculated from related attribute values.


---

## Physical ERD Symbols

The symbols below are used at the most granular level of ERDs: physical data models, although some elements are also used for logical data models.

- Tables are another way of representing entities.
- Fields represent attributes of the entity.
- Keys are one way to categorize attributes. A primary key is an attribute or combination of attributes that uniquely identifies one and only one instance of an entity. The primary key becomes a foreign key in any entity type to which it's related through a one-to-one or one-to-many relationship.
- Types may refer to the type of data associated with the corresponding field in a table. Types can also refer to entity types, which describe the structure of an entity; e.g., a book's entity types are author, title, and published date.

| Entity |
|---|
| Field |
| Field |
| Field |

| Entity | |
|---|---|
| Key | Field |
| Key | Field |
| Key | Field |

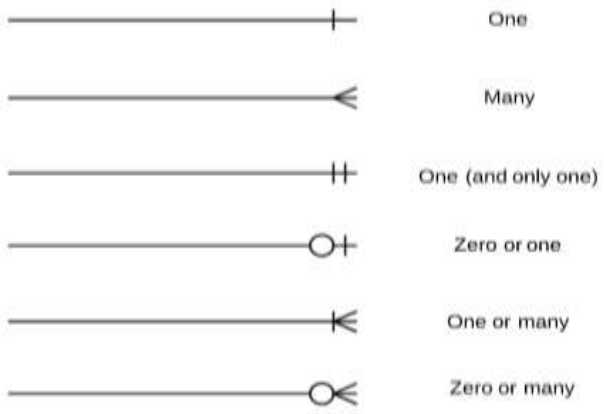
| Entity | |
|---|---|
| Field | Type |
| Field | Type |
| Field | Type |

| Entity | | |
|---|---|---|
| Key | Field | Type |
| Key | Field | Type |
| Key | Field | Type |

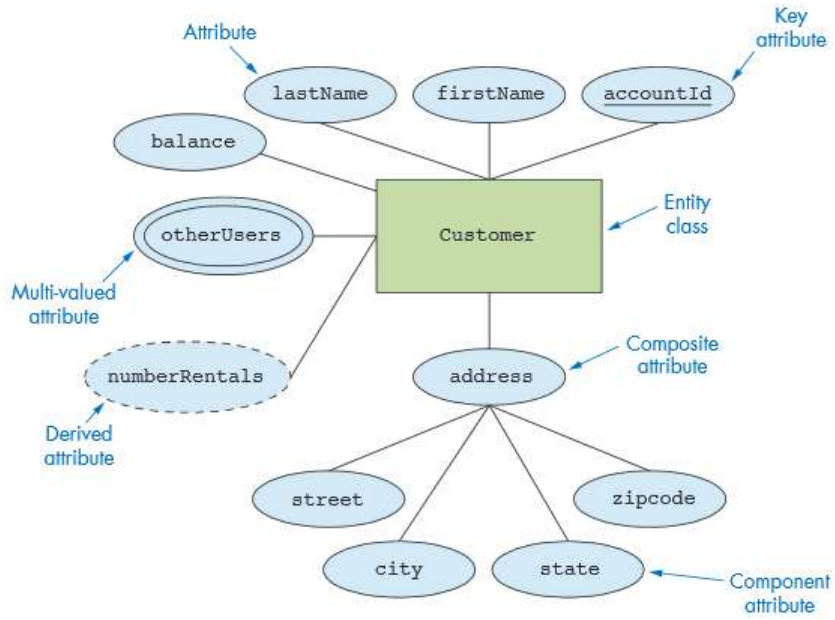
---

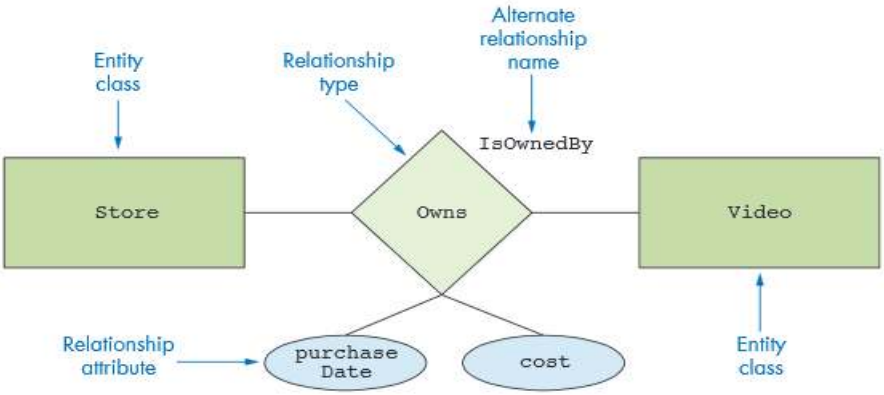
**ERD Notation**

- Relationships illustrate an association between two tables. In the physical data model, relationships are represented by stylized lines.
- Cardinality and ordinality, respectively, refer to the maximum number of times an instance in one entity can be associated with instances in the related entity, and the minimum number of times an instance in one entity can be associated with an instance in the related entity. Cardinality and ordinality are represented by the styling of a line and its endpoint, as denoted by the chosen notation style.



Examples



# CHAPTER 6: NORMALIZATION

## Introduction and meaning of Database normalization

Database normalization is the process of organizing the fields and tables of a relational database to minimize redundancy. Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them.

Normalization of data can be defined as a process during which the existing tables of a database are tested to find certain data dependency between the column and the rows or normalizing of data can be referred to a formal technique of making preliminary data structures into an easy to maintain and make efficient data structure

With data normalization any table dependency detected, the table is restructured into multiple tables (two tables) which eliminate any column dependency. Incase data dependency is still exhibited the process is repeated till such dependency are eliminated. The process of eliminating data redundancy is based upon a theory called functional dependency

## Importance of normalization

- It highlights constraints and dependency in the data and hence aid the understanding the nature of the data
- Normalization controls data redundancy to reduce storage requirement and standard maintenance
- Normalization provide unique identification for records in a database
- Each stage of normalization process eliminate a particular type of undesirable dependency
- Normalization permits simple data retrieval in response to reports and queries
- The third normalization form produces well designed database which provides a higher degree of independency
- Normalization helps define efficient data structures
- Normalized data structures are used for file and database design
- Normalization eliminate unnecessary dependency relationship within a database file

## Forms of normalization/Normalization rules

### First normal form (1NF)

Refers to the first step where preliminary data structures are transforming into the first normal form by eliminating any repeating sets of data elements. A relation table is said to be on the first normal form, if and only if it contains no repeating groups that is it has no repeated value for a particular attribute with a single record. Any repeated group of attribute is isolated to form a new relation. In other words first normal form (1nf) means that a table has no multiple value attribute or composite attribute, In the 1nf, each column holds one attribute and each row holds a single occurrence of the entity

### Second normal form (2NF)

2nf concentrated on records with concatenated keys, they check the non key attribute for dependency on the entire key, and any data element that dependent only on part of the key is moved to a new entity

### Third normal form (3NF)

All data element in the third normal form must be a function of the key. To reach the 3nf, you need to review the structure's non-key data elements and identify any data element dependent on an attribute other than the key, if there is all these data elements should be moved to a new entity

### Fourth normal form (4NF)

In data normalization, the fourth normal form deals with data element with issues of multi-value dependency (when one attributes determine another attribute sets). A relation is said to be in the 4nf formal form if and if only all existing multi-value dependency is converted into functional dependency

### Fifth normal form (5NF)

Here is where the join dependency is removed, the 5nf is also known as the projection join normal form(PJNF), and refers to the separation of one relation into any sub-relations or having sub-relations into one relation and can produce join dependencies

## Performing Normalization by example

While designing a database out of an entity–relationship model, the main problem existing in that "raw" database is redundancy. Redundancy is storing the same data item in more one place. A redundancy creates several problems like the following:

1. Extra storage space: storing the same data in many places takes large amount of disk space.
2. Entering same data more than once during data insertion.
3. Deleting data from more than one place during deletion.
4. Modifying data in more than one place.
5. Anomalies may occur in the database if insertion, deletion, modification etc are no done properly. It creates inconsistency and unreliability in the database.

To solve this problem, the "raw" database needs to be normalized. This is a step by step process of removing different kinds of redundancy and anomaly at each step. At each step a specific rule is followed to remove specific kind of impurity in order to give the database a slim and clean look.

#### Un-Normalized Form (UNF)

If a table contains non-atomic values at each row, it is said to be in UNF. An atomic value is something that can not be further decomposed. A non-atomic value, as the name suggests, can be further decomposed and simplified. Consider the following table:

| Emp-Id | Emp-Name | Month | Sales | Bank-Id | Bank-Name |
|---|---|---|---|---|---|
| E01 | AA | Jan | 1000 | B01 | SBI |
| | | Feb | 1200 | | |
| | | Mar | 850 | | |
| E02 | BB | Jan | 2200 | B02 | UTI |
| | | Feb | 2500 | | |
| E03 | CC | Jan | 1700 | B01 | SBI |
| | | Feb | 1800 | | |
| | | Mar | 1850 | | |
| | | Apr | 1725 | | |

In the sample table above, there are multiple occurrences of rows under each key Emp-Id. Although considered to be the primary key, Emp-Id cannot give us the unique identification facility for any single row. Further, each primary key points to a variable length record (3 for E01, 2 for E02 and 4 for E03).

### First Normal Form (1NF)

A relation is said to be in 1NF if it contains no non-atomic values and each row can provide a unique combination of values. The above table in UNF can be processed to create the following table in 1NF.

| Emp-Id | Emp-Name | Month | Sales | Bank-Id | Bank-Name |
|---|---|---|---|---|---|
| E01 | AA | Jan | 1000 | B01 | SBI |
| E01 | AA | Feb | 1200 | B01 | SBI |
| E01 | AA | Mar | 850 | B01 | SBI |
| E02 | BB | Jan | 2200 | B02 | UTI |
| E02 | BB | Feb | 2500 | B02 | UTI |
| E03 | CC | Jan | 1700 | B01 | SBI |
| E03 | CC | Feb | 1800 | B01 | SBI |
| E03 | CC | Mar | 1850 | B01 | SBI |
| E03 | CC | Apr | 1725 | B01 | SBI |

As you can see now, each row contains unique combination of values. Unlike in UNF, this relation contains only atomic values, i.e. the rows can not be further decomposed, so the relation is now in 1NF.

### Second Normal Form (2NF)

A relation is said to be in 2NF f if it is already in 1NF and each and every attribute fully depends on the primary key of the relation. Speaking inversely, if a table has some attributes which is not dependant on the primary key of that table, then it is not in 2NF.

Let us explain. Emp-Id is the primary key of the above relation. Emp-Name, Month, Sales and Bank-Name all depend upon Emp-Id. But the attribute Bank-Name depends on Bank-Id, which is not the primary key of the table. So the table is in 1NF, but not in 2NF. If this position can be removed into another related relation, it would come to 2NF.

| Emp-Id | Emp-Name | Month | Sales | Bank-Id |
|---|---|---|---|---|
| E01 | AA | JAN | 1000 | B01 |
| E01 | AA | FEB | 1200 | B01 |
| E01 | AA | MAR | 850 | B01 |
| E02 | BB | JAN | 2200 | B02 |
| E02 | BB | FEB | 2500 | B02 |
| E03 | CC | JAN | 1700 | B01 |
| E03 | CC | FEB | 1800 | B01 |
| E03 | CC | MAR | 1850 | B01 |
| E03 | CC | APR | 1726 | B01 |

| Bank-Id | Bank-Name |
|---|---|
| B01 | SBI |
| B02 | UTI |

After removing the portion into another relation we store lesser amount of data in two relations without any loss information. There is also a significant reduction in redundancy.

### Third Normal Form (3NF)

A relation is said to be in 3NF, if it is already in 2NF and there exists no transitive dependency in that relation. Speaking inversely, if a table contains transitive dependency, then it is not in 3NF, and the table must be split to bring it into 3NF.

What is a transitive dependency? Within a relation if we see
A → B [B depends on A]
And
B → C [C depends on B]
Then we may derive
A → C[C depends on A]

Such derived dependencies hold well in most of the situations. For example if we have
Roll → Marks
And
Marks → Grade
Then we may safely derive
Roll → Grade.

This third dependency was not originally specified but we have derived it.

The derived dependency is called a transitive dependency when such dependency becomes improbable. For example we have been given
Roll → City
And
City → STDCode

If we try to derive Roll → STDCode it becomes a transitive dependency, because obviously the STDCode of a city cannot depend on the roll number issued by a school or college. In such a case the relation should be broken into two, each containing one of these two dependencies:
Roll → City
And
City → STD code

### Boyce-Code Normal Form (BCNF)

A relationship is said to be in BCNF if it is already in 3NF and the left hand side of every dependency is a candidate key. A relation which is in 3NF is almost always in BCNF. These

could be same situation when a 3NF relation may not be in BCNF the following conditions are found true.

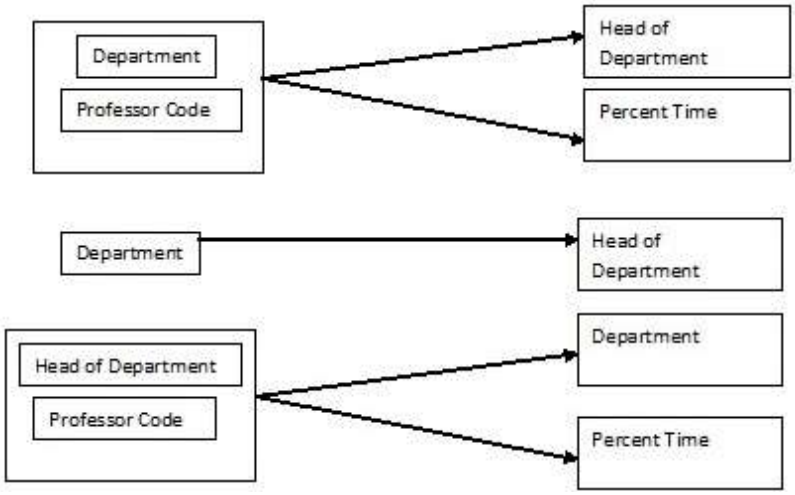
1. The candidate keys are composite.
2. There are more than one candidate keys in the relation.
3. There are some common attributes in the relation.

| Professor Code | Department | Head of Dept. | Percent Time |
|---|---|---|---|
| P1 | Physics | Ghosh | 50 |
| P1 | Mathematics | Krishnan | 50 |
| P2 | Chemistry | Rao | 25 |
| P2 | Physics | Ghosh | 75 |
| P3 | Mathematics | Krishnan | 100 |

Consider, as an example, the above relation. It is assumed that:

1. A professor can work in more than one department
2. The percentage of the time he spends in each department is given.
3. Each department has only one Head of Department.

The relation diagram for the above relation is given as the following:



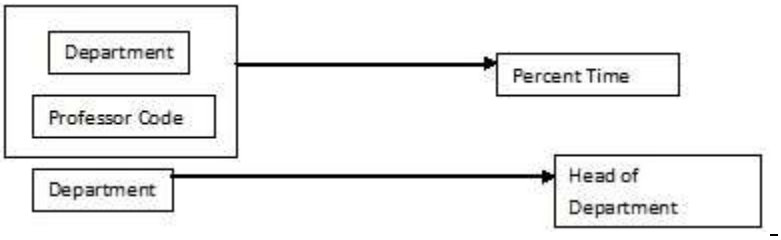

The given relation is in 3NF. Observe, however, that the names of Dept. and Head of Dept. are duplicated. Further, if Professor P2 resigns, rows 3 and 4 are deleted. We lose the information that Rao is the Head of Department of Chemistry.

The normalization of the relation is done by creating a new relation for Dept. and Head of Dept. and deleting Head of Dept. form the given relation. The normalized relations are shown in the following.

| Professor Code | Department | Percent Time |
|---|---|---|
| P1 | Physics | 50 |
| P1 | Mathematics | 50 |
| P2 | Chemistry | 25 |
| P2 | Physics | 75 |
| P3 | Mathematics | 100 |

| Department | Head of Dept. |
|---|---|
| Physics | Ghosh |
| Mathematics | Krishnan |
| Chemistry | Rao |

See the dependency diagrams for these new relations.




**Fourth Normal Form (4NF)**

When attributes in a relation have multi-valued dependency, further Normalization to 4NF and 5NF are required. Let us first find out what multi-valued dependency is.

A multi-valued dependency is a typical kind of dependency in which each and every attribute within a relation depends upon the other, yet none of them is a unique primary key.

We will illustrate this with an example. Consider a vendor supplying many items to many projects in an organization. The following are the assumptions:

1. A vendor is capable of supplying many items.
2. A project uses many items.
3. A vendor supplies to many projects.
4. An item may be supplied by many vendors.

A multi valued dependency exists here because all the attributes depend upon the other and yet none of them is a primary key having unique value.

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

The given relation has a number of problems. For example:

1. If vendor V1 has to supply to project P2, but the item is not yet decided, then a row with a blank for item code has to be introduced.
2. The information about item I1 is stored twice for vendor V3.

Observe that the relation given is in 3NF and also in BCNF. It still has the problem mentioned above. The problem is reduced by expressing this relation as two relations in the Fourth Normal Form (4NF). A relation is in 4NF if it has no more than one independent multi valued dependency or one independent multi valued dependency with a functional dependency.

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

**Vendor-Supply**

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

**Vendor-Project**

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

### Fifth Normal Form (5NF)

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

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

Let us finally summarize the normalization steps we have discussed so far.

| Input Relation | Transformation | Output Relation |
|---|---|---|
| All Relations | Eliminate variable length record. Remove multi-attribute lines in table. | 1NF |
| 1NF Relation | Remove dependency of non-key attributes on part of a multi-attribute key. | 2NF |
| 2NF | Remove dependency of non-key attributes on other non-key attributes. | 3NF |
| 3NF | Remove dependency of an attribute of a multi attribute key on an attribute of another (overlapping) multi-attribute key. | BCNF |
| BCNF | Remove more than one independent multi-valued dependency from relation by splitting relation. | 4NF |
| 4NF | Add one relation relating attributes with multi-valued dependency. | 5NF |

# CHAPTER 7: QUERYING A DATABASE

## Definition and Meaning of Database Query:

Queries are the primary mechanism for retrieving information from a database and consist of questions presented to the database in a predefined format. Many database management systems use the Structured Query Language (SQL) standard query format.

## SQL Features of Query language (SQL)

| General Features | |
|---|---|
| Multi-platform Support | Supports all major DBMSs from a single interface. Ability to use the tools on all supported platforms from a single license. |
| Embarcadero® AppWave™ | Enables centralized license management and tool deployment |
| Unicode Support | Full Unicode character support throughout the application |
| Intuitive Interface | Automates common and repetitive tasks with easy-to-use SQL editors and wizards |
| SQL Scripting and Editing | |
| Visual Query Builder | Constructs even the most complicated SQL statements with point-and-click ease |
| Code Templates | Eliminates the need to memorize and type SQL syntax |
| SQL Editor | Code folding, code collapse/roll-up, syntax coloring, hot key assignments, configurable auto replace of objects, bind variable support, selective statement execution |
| Context-sensitive DMBS Actions | DBMS actions, such as Extract and Drop, are available directly in the context menu of the appropriate tokens in the SQL editor |
| Debugging, Performance Optimization | |
| Code Analyst | Performs detailed response time analysis on the execution of stored procedures and functions |

| | |
|---|---|
| SQL Debugger | Debugs programmable objects such as stored procedures, functions, packages, and triggers. Available for DB2 for LUW, Oracle, SQL Server, and Sybase |
| SQL Profiler | Captures metrics of various PL/SQL programmable objects on Oracle 8.1.5 and higher. |
| Developer Features | |
| Advanced Code Assist | Lists context-sensitive suggestions as you type (e.g., tables, columns, procedures, functions, and code templates) and is available offline |
| Code Formatting and Profiles | Code folding, syntax coloring, comment toggling, and other auto formatting features make it easy to read, navigate, and edit large SQL files. Customize and share various SQL formatting options by creating SQL formatting profiles |
| Context-sensitive DBMS Actions | DBMS actions, such as Extract and Drop, are available directly in the context menu of the appropriate tokens in the SQL editor |
| Syntax & Semantic Validation | Validate SQL files and flags all DBMS-specific parser violations or references to objects not found in the target database |
| Quick Fixes | Real-time parsing provides code quality suggestions for improving SQL performance as you type |
| SQL Debugging | Debug Java, step seamlessly into SQL (i.e. stored procedure) and back into Java again – true system-wide, round-trip debugging |
| Data Governance* | |
| Inline Metadata | Gives users real-time metadata visibility in the SQL IDE and will gain valuable context in SQL query development with awareness of sensitive data. Examples of metadata attributes are: table descriptions, PII, data governance policy information, etc. |
| Centralized Datasource Repository | Provides the functionality for users to work from a common, centralized datasource repository |

## SQL Commands and categories:

SQL commands are instructions, coded into SQL statements, which are used to communicate with the database to perform specific tasks, work, functions and queries with data.

SQL commands can be used not only for searching the database but also to perform various other functions like, for example, you can *create tables*, add data to tables, or modify data, drop the table, set permissions for users.

SQL commands are grouped into four major categories depending on their functionality:

- Data Definition Language (DDL) - These SQL commands are used for creating, modifying, and dropping the structure of database objects. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.
- Data Manipulation Language (DML) - These SQL commands are used for storing, retrieving, modifying, and deleting data.
  These Data Manipulation Language commands are: SELECT INSERT, UPDATE, and DELETE.
- Transaction Control Language (TCL) - These SQL commands are used for managing changes affecting the data. These commands are COMMIT, ROLLBACK, and SAVEPOINT.
- Data Control Language (DCL) - These SQL commands are used for providing security to database objects. These commands are GRANT and REVOKE.

## SQL statements/Queries design and interrogation of database

### SQL statement design

✓ **The CREATE DATABASE Statement**

The CREATE DATABASE statement is used to create a database.

**SQL CREATE DATABASE Syntax**

```
CREATE DATABASE database_name
```

**CREATE DATABASE Example**

Now we want to create a database called "my_db".

We use the following CREATE DATABASE statement:

```
CREATE DATABASE my_db
```

Database tables can be added with the CREATE TABLE statement.

## The CREATE TABLE Statement

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

### SQL CREATE TABLE Syntax

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
....
)
```

## CREATE TABLE Example

Now we want to create a table called "Persons" that contains five columns: P_Id, LastName, FirstName, Address, and City.

We use the following CREATE TABLE statement:

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

The P_Id column is of type int and will hold a number. The LastName, FirstName, Address, and City columns are of type varchar with a maximum length of 255 characters.

The empty "Persons" table will now look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| | | | | |

The empty table can be filled with data with the INSERT INTO statement.

✓ **Describe command**

The describe command gives you a list of all the data fields used in your database table. In the example, you can see that the table named test in the sales data database keeps track of four fields: name, description, num, and date_modified.

```
mysql> describe test;
+---------------+-------------+------+-----+------------+----------------+
| Field         | Type        | Null | Key | Default    | Extra          |
+---------------+-------------+------+-----+------------+----------------+
| num           | int(11)     |      | PRI | NULL       | auto_increment |
| date_modified | date        |      | MUL | 0000-00-00 |                |
| name          | varchar(50) |      | MUL |            |                |
| description   | varchar(75) | YES  |     | NULL       |                |
+---------------+-------------+------+-----+------------+----------------+
6 rows in set (0.00 sec)
```

## SQL database interaction

✓ **The INSERT INTO Statement**

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

***SQL INSERT INTO Syntax***

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

The first form doesn't specify the column names where the data will be inserted, only their values:

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

The second form specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3,...)
```

```
VALUES (value1, value2, value3,...)
```

---

**SQL INSERT INTO Example**

We have the following "Persons" table:

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

Now we want to insert a new row in the "Persons" table.

We use the following SQL statement:

```
INSERT INTO Persons
VALUES (4,'Nilsen', 'Johan', 'Bakken 2', 'Stavanger')
```

The "Persons" table will now look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |

---

### Insert Data Only in Specified Columns

It is also possible to only add data in specific columns.

The following SQL statement will add a new row, but only add data in the "P_Id", "LastName" and the "FirstName" columns:

```
INSERT INTO Persons (P_Id, LastName, FirstName)
VALUES (5, 'Tjessem', 'Jakob')
```

The "Persons" table will now look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |
| 5 | Tjessem | Jakob | | |

## ✓ The UPDATE Statement

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

SQL UPDATE Syntax

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

**Note:** Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

**SQL UPDATE Example**

The "Persons" table:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |
| 5 | Tjessem | Jakob | | |

Now we want to update the person "Tjessem, Jakob" in the "Persons" table.

We use the following SQL statement:

```
UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
WHERE LastName='Tjessem' AND FirstName='Jakob'
```

The "Persons" table will now look like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |
| 4 | Nilsen | Johan | Bakken 2 | Stavanger |
| 5 | Tjessem | Jakob | Nissestien 67 | Sandnes |

**SQL UPDATE Warning**

Be careful when updating records. If we had omitted the WHERE clause in the example above, like this:

```
UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
```

The "Persons" table would have looked like this:

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Nissestien 67 | Sandnes |
| 2 | Svendson | Tove | Nissestien 67 | Sandnes |
| 3 | Pettersen | Kari | Nissestien 67 | Sandnes |
| 4 | Nilsen | Johan | Nissestien 67 | Sandnes |
| 5 | Tjessem | Jakob | Nissestien 67 | Sandnes |

## More SQL commands

1. **Create a database on the sql server.**

   mysql> create database [databasename];

2. **List all databases on the sql server.**

   mysql> show databases;

3. **Switch to a database.**

   mysql> use [db name];

4. **To see all the tables in the db.**

   mysql> show tables;

5. **To see database's field formats.**

mysql> describe [table name];

6. **To delete a db.**

   mysql> drop database [database name];

7. **To delete a table.**

   mysql> drop table [table name];

8. **Show all data in a table.**

   mysql> SELECT * FROM [table name];

9. **Returns the columns and column information pertaining to the designated table.**

   mysql> show columns from [table name];

10. **Show certain selected rows with the value "whatever".**

    mysql> SELECT * FROM [table name] WHERE [field name] = "whatever";

11. **Show all records containing the name "Bob" AND the phone number '3444444'.**

    mysql> SELECT * FROM [table name] WHERE name = "Bob" AND phone_number = '3444444';

12. **Show all records not containing the name "Bob" AND the phone number '3444444' order by the phone_number field.**

    mysql> SELECT * FROM [table name] WHERE name != "Bob" AND phone_number = '3444444' order by phone_number;

13. **Show all records starting with the letters 'bob' AND the phone number '3444444'.**

    mysql> SELECT * FROM [table name] WHERE name like "Bob%" AND phone_number = '3444444';

14. **Show all records starting with the letters 'bob' AND the phone number '3444444' limit to records 1 through 5.**

    mysql> SELECT * FROM [table name] WHERE name like "Bob%" AND phone_number = '3444444' limit 1,5;

**15. Use a regular expression to find records. Use "REGEXP BINARY" to force case-sensitivity. This finds any record beginning with a.**

mysql> SELECT * FROM [table name] WHERE rec RLIKE "^a";

**16. Show unique records.**

mysql> SELECT DISTINCT [column name] FROM [table name];

**17. Show selected records sorted in an ascending (asc) or descending (desc).**

mysql> SELECT [col1],[col2] FROM [table name] ORDER BY [col2] DESC;

**18. Return number of rows.**

mysql> SELECT COUNT(*) FROM [table name];

**19. Sum column.**

mysql> SELECT SUM(*) FROM [table name];

**20. Join tables on common columns.**

mysql> select lookup.illustrationid, lookup.personid,person.birthday from lookup left join person on lookup.personid=person.personid=statement to join birthday in person table with primary illustration id;

**21. To update info already in a table.**

mysql> UPDATE [table name] SET Select_priv = 'Y',Insert_priv = 'Y',Update_priv = 'Y' where [field name] = 'user';

**22. Delete a row(s) from a table.**

mysql> DELETE from [table name] where [field name] = 'whatever';

**23. Delete a column.**

mysql> alter table [table name] drop column [column name];

**24. Add a new column to db.**

mysql> alter table [table name] add column [new column name] varchar (20);

**25. Change column name.**

mysql> alter table [table name] change [old column name] [new column name] varchar (50);

**26. Make a unique column so you get no dupes.**

mysql> alter table [table name] add unique ([column name]);

**27. Make a column bigger.**

mysql> alter table [table name] modify [column name] VARCHAR(3);

**28. Delete unique from table.**

mysql> alter table [table name] drop index [colmn name];

**29. Create Table Example 1.**

mysql> CREATE TABLE [table name] (firstname VARCHAR(20), middleinitial VARCHAR(3), lastname VARCHAR(35),suffix VARCHAR(3),officeid VARCHAR(10),userid VARCHAR(15),username VARCHAR(8),email VARCHAR(35),phone VARCHAR(25), groups VARCHAR(15),datestamp DATE,timestamp time,pgpemail VARCHAR(255));

**30. Create Table Example 2.**

mysql> create table [table name] (personid int(50) not null auto_increment primary key,firstname varchar(35),middlename varchar(50),lastnamevarchar(50) default 'bato');

# CHAPTER 8: FUNCTIONS OF DBMS

## Introduction to DBMS functions

A software that handles the storage, retrieval, and updating of data in a computer system.

A database management system (DBMS) is a collection of programs that enables you to store, modify, and extract information from a database. There are many different types of DBMSs, ranging from small systems that run on personal computers to huge systems that run on mainframes. The following are examples of database applications:

- Computerized library systems
- Automated teller machines
- Flight reservation systems
- Computerized parts inventory systems

**Importance of DBMS:**

A database management system is important because it manages data efficiently and allows users to perform multiple tasks with ease.

A database management system stores, organizes and manages a large amount of information within a single software application. Use of this system increases efficiency of business operations and reduces overall costs.

Handling multiple types of data. Some of the data that are easily managed with this type of system include: employee records, student information, payroll, accounting, project management, inventory and library books. These systems are built to be extremely versatile.

## DBMS Functions

There are several functions that a DBMS performs to ensure data integrity and consistency of data in the database. The ten functions in the DBMS are: data dictionary management, data storage management, data transformation and presentation, security management, multiuser access control, backup and recovery management, data integrity management, database access languages and application programming interfaces, database communication interfaces, and transaction management.

### 1. Data Dictionary Management

Data Dictionary is where the DBMS stores definitions of the data elements and their relationships (metadata). The DBMS uses this function to look up the required data component structures and relationships. When programs access data in a database they are basically going through the DBMS. This function removes structural and data dependency and provides the user with data abstraction. In turn, this makes things a lot easier on the end user. The Data Dictionary is often hidden from the user and is used by Database Administrators and Programmers.are u mad

**2. Data Storage Management**

This particular function is used for the storage of data and any related data entry forms or screen definitions, report definitions, data validation rules, procedural code, and structures that can handle video and picture formats. Users do not need to know how data is stored or manipulated. Also involved with this structure is a term called performance tuning that relates to a database's efficiency in relation to storage and access speed.

**3. Data Transformation and Presentation**

This function exists to transform any data entered into required data structures. By using the data transformation and presentation function the DBMS can determine the difference between logical and physical data formats.

**4. Security Management**

This is one of the most important functions in the DBMS. Security management sets rules that determine specific users that are allowed to access the database. Users are given a username and password or sometimes through biometric authentication (such as a fingerprint or retina scan) but these types of authentication tend to be more costly. This function also sets restraints on what specific data any user can see or manage.

**5. Multiuser Access Control**

Data integrity and data consistency are the basis of this function. Multiuser access control is a very useful tool in a DBMS, it enables multiple users to access the database simultaneously without affecting the integrity of the database.

**6. Backup and Recovery Management**

Backup and recovery is brought to mind whenever there is potential outside threats to a database. For example if there is a power outage, recovery management is how long it takes to recover the database after the outage. Backup management refers to the data safety and integrity; for example backing up all your mp3 files on a disk.

**7. Data Integrity Management**

The DBMS enforces these rules to reduce things such as data redundancy, which is when data is stored in more than one place unnecessarily, and maximizing data consistency, making sure database is returning correct/same answer each time for same question asked.

**8. Database Access Languages and Application Programming Interfaces**

A query language is a nonprocedural language. An example of this is SQL (structured query language). SQL is the most common query language supported by the majority of DBMS vendors. The use of this language makes it easy for user to specify what they want done without the headache of explaining how to specifically do it.

**9. Database Communication Interfaces**

This refers to how a DBMS can accept different end user requests through different network environments. An example of this can be easily related to the internet. A DBMS can provide access to the database using the Internet through Web Browsers (Mozilla Firefox, Internet Explorer, Netscape).

### *10.* Transaction Management

This refers to how a DBMS must supply a method that will guarantee that all the updates in a given transaction are made or not made.All transactions must follow what is called the ACID properties.

**A – Atomicity:** states a transaction is an indivisible unit that is either performed as a whole and not by its parts, or not performed at all.It is the responsibility of recovery management to make sure this takes place.

**C – Consistency:**A transaction must alter the database from one constant state to another constant state.

**I – Isolation:** Transactions must be executed independently of one another.Part of a transaction in progress should not be able to be seen by another transaction.

**D – Durability:** A successfully completed transaction is recorded permanently in the database and must not be lost due to failures**.**

**More specifically, a DBMS provides the following functions:**

- Concurrency: concurrent access (meaning 'at the same time') to the same database by multiple users
- Security: security rules to determine access rights of users
- Backup and recovery: processes to back-up the data regularly and recover data if a problem occurs
- Integrity: database structure and rules improve the integrity of the data
- Data descriptions: a data dictionary provides a description of the data

Within an organization, the development of the database is typically controlled by **database administrators (DBAs)** and other specialists. This ensures the database structure is efficient and reliable.

# CHAPTER 9: EMERGING TRENDS

## Trends in Database Management

Concepts in database management hardly fall in the category of come-and-go, as the cost of shifting between technical approaches overwhelms producers, managers, and designers. However, there are several trends in database management, and knowing how to take advantage of them will benefit your organization. Following are the some of the current trends:

**1. Databases that bridge SQL/NoSQL**
The latest trends in database products are those that don't simply embrace a single database structure. Instead, the databases bridge SQL and NoSQL, giving users the best capabilities offered by both. This includes products that allow users to access a NoSQL database in the same way as a relational database, for example.

**2. Databases in the cloud/Platform as a Service**
As developers continue pushing their enterprises to the cloud, organizations are carefully weighing the trade-offs associated with public versus private. Developers are also determining how to combine cloud services with existing applications and infrastructure. Providers of cloud service offer many options to database administrators. Making the move towards the cloud doesn't mean changing organizational priorities, but finding products and services that help your group meet its goals.

**3. Automated management**
Automating database management is another emerging trend. The set of such techniques and tools intend to simplify maintenance, patching, provisioning, updates and upgrades — even project workflow. However, the trend may have limited usefulness since database management frequently needs human intervention.

**4. An increased focus on security**
While not exactly a trend given the constant focus on data security, recent ongoing retail database breaches among US-based organizations show with ample clarity the importance for database administrators to work hand-in-hand with their IT security colleagues to ensure all enterprise data remains safe. Any organization that stores data is vulnerable.
Database administrators must also work with the security team to eliminate potential internal weaknesses that could make data vulnerable. These could include issues related to network privileges, even hardware or software misconfigurations that could be misused, resulting in data leaks.

**5. In-memory databases**
Within the data warehousing community there are similar questions about columnar versus row-based relational tables; the rise of in-memory databases, the use of flash or solid-state disks (which also applies within transaction processing), clustered versus no-clustered solutions and so on.

6. **Big Data**
To be clear, big data does not necessarily mean lots of data. What it really refers to is the ability to process any type of data: what is typically referred to as semi-structured and unstructured data as well as structured data. Current thinking is that these will typically live alongside conventional solutions as separate technologies, at least in large organisations, but this will not always be the case.
Integrating Trends
Projects involving databases should not be viewed and appreciated solely on how they adhere to these trends. Ideally, each tool or process available should merge in some meaningful way with existing operations. It is important to look of these trends as items that can coincide: enhancing security and moving to the cloud coexist?

## The Top Challenges and Solutions of Database Management

No matter what field you work in, there will be changes over time. As technology becomes more and more advanced, everyone from doctors to politicians and athletes must learn to use these changes to their advantage. While other professions have encountered these changes, few have experienced them on the same level as database administrators.

Thirty years after the computerization of databases, the Internet has lead to an exponential growth within the industry – whether indirectly or directly, everything that compiles data uses a database. Recent times have proven to be an exceptional period of the production and capturing of a nearly overwhelming amount of data. This has obviously created opportunities for businesses to gain visibility into their customers and industry, but it has also created many challenges in database management.

### Database Management Problems

- Data Integration from Various Sources – With the advancement of smartphones, new mobile applications, and the Internet of Things, businesses must be able to have their data adapt accordingly. These varying types of data and sources cause a typical data center of today to contain patchwork for data management technologies. The management techniques have become more diverse than ever.
- Public and Private Data Security – In today's digital world, security is the most prevalent concern. Businesses must be able to ensure that every bit of their data remains safe and at limited risk of exposure from hackers or leaks. Database breaches of highly sensitive information have led to the destroyed reputation of businesses. It is up to the manager of the database to ensure that the data is fully secured at all times.
- The Management of Cloud-Based Databases – In recent years, the Cloud has become one of the biggest terms in the tech community. Both businesses and consumers want to be able to access their data from database from the cloud or from a cloud database provider's servers in addition to the standard on-premises mode of deployment. Cloud computing enables users to effectively allocate resources, optimize scaling, and allow for high availability. Handling database that run on the cloud and on-premises is yet another challenge for database managers.
- The Growth of Structured and Unstructured Data – The amount of data that has being both created and collected has been growing at an unprecedented rate for years. Those who deal with analytics may be excited by the promise of insight and business intelligence that comes

from big data, but those who manage databases face the challenges that come along with managing overall growth and data types from an increasing number of database platforms.

## Database Management Solutions

There are four main areas to think about when thinking about approaching these database problems. The following are a few things to consider as solutions:

- Data Strategy
    - What kind of data is important and what kind of performance should be achieved? What data needs to be protected and what should be analyzed?
    - How much historical data must be accumulated? What does this mean for capacity planning and disk space?
    - Can you monetize on your data? Which data needs to be aggregated or correlated to provide the necessary insights into the business?
- Database Support
    - You must consider that moving to the cloud does not guarantee data backup and security. This is something that must still be managed with 24/7 monitoring and coverage.
    - Are the right personnel members with the necessary skill sets always available?
- Backup Strategy
    - Do you have the right kind of backup retention available?
    - Have you determined the necessary backup frequency to determine the Recovery Point Objective (RPO)?
    - Have you determined the Recovery Time Objective (RTO) due to high availability requirements?
- Security Strategy
    - How will external and internal security be handled? Who can access what?
    - What kind of data access policies should be in place?
    - How are regulatory requirements handled?
    - In the event of a hack, breach, or leak, how will data exposure be handled?