DATABASE MANAGEMENT SYSTEMS
LECTURE NOTES

2rd Class, CS Dept.
1st Semester

UNIT-ONE

Introduction to Database Management System

As the name suggests, the database management system consists of two parts. They are:

1. ***Database*** and
2. ***Management System***

What is a Database?

To find out what database is, we have to start from data, which is the basic building block of any DBMS.

- **Data**: *Facts*, *figures*, *statistics* etc. **having no particular meaning** (e.g. 01, ABC, 19 etc).
- **Record**: *Collection of related data items*, **e.g**. in the above example the three data items had no meaning. However, if we organize them in the following way, then they collectively represent ***meaningful information***.

Roll	Name	Age
01	ABC	19

- **Table** or **Relation**: Collection of related records.

Roll	Name	Age
01	ABC	19
02	DEF	22
03	XYZ	28

  - The columns of this relation are called **Fields**, **Attributes** or **Domains**.
  - The rows are called **Tuples** or **Records**.

⇒ **Database**: Collection of related relations.

Consider the following collection of tables:

T1

Roll	Name	Age
01	ABC	19
02	DEF	22
03	XYZ	28

T2

Roll	Address
01	**KOL**
02	**DEL**
03	**MUM**

T3

Roll	Year
01	**I**
02	**II**
03	**I**

T4

Year	Hostel
I	H1
II	H2

We now have a collection of ***four*** tables. They can be called a "**related collection**" because we can clearly find out that there are some common attributes existing in a selected pair of tables. Because of these common attributes, we may *combine* the data of two or more tables together to find out the complete details of a student.

Questions like "Which **hostel** does the ***youngest student*** live in?" can be answered now, although ***Age*** and ***Hostel*** attributes are in different tables.

A database in a DBMS could be viewed by lots of different people with different responsibilities.

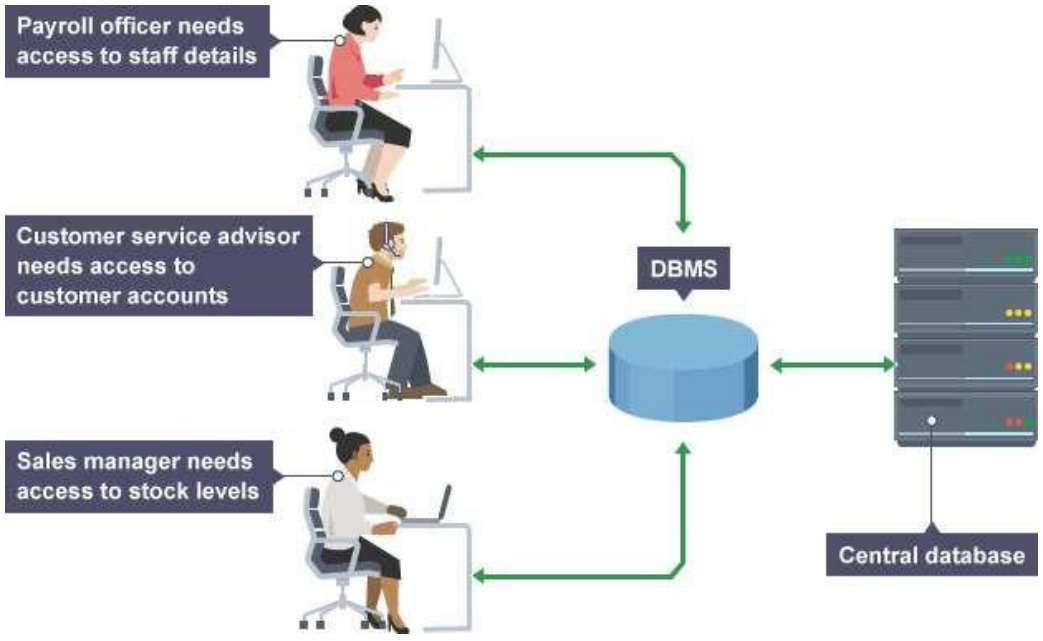


Figure 1.1: Employees are accessing Data through DBMS

For example, within a company there are different ***departments***, as well as ***customers***, who each need to see different kinds of data. Each ***employee*** in the company will have different levels of ***access*** to the database with their own customized **front-end** application.

In a database, data is organized strictly in ***row*** and ***column*** format. The rows are called ***Tuple*** or ***Record***.

The data items within one row may belong to different data types.

On the other hand, the columns are often called **Domain** or **Attribute**. All the data items within a single attribute are of the same data type.

What is Management System?

- A **database-management system** (**DBMS**) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a **database**. The collection of data, usually referred to as the **database**, *contains information relevant to an enterprise*.
- **The primary goal** of a DBMS is to provide a way to *store* and *retrieve* database information that is both *convenient* and *efficient*. By **data,** we mean known facts that can be recorded and that have *implicit meaning*.

The management system is important, why? Because without the existence of some kind of rules and regulations it is not possible to maintain the database. We have to select the particular attributes, which should be included in a particular table; the common attributes to create relationship between two tables; if a new record has to be inserted or deleted then which tables should have to be handled etc. These issues must be resolved by having some kind of rules to follow in order to maintain the ***integrity*** of the database.

- **Database systems** are ***designed*** to manage large bodies of information.
- **Management of data** involves both defining structures for storage of information and providing mechanisms for the manipulation of information.

In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

Because information is so important in most organizations, computer scientists have developed a large body of concepts and techniques for managing data. This chapter briefly introduces the principles of database systems.

Database Management System (DBMS) and Its Applications:

A **Database management system** is a computerized record-keeping system. It is a repository or a container for collection of computerized data files. The overall purpose of DBMS is to allow the users to *define*, *store*, *retrieve* and *update* the information contained in the database on demand. Information can be anything that is of significance to an individual or organization.

Databases touch all aspects of our lives. Some of the major areas of application are as follows:

1. Banking
2. Airlines
3. Universities
4. Manufacturing and selling
5. Human resources

- **Enterprise Information**
  - ***Sales***: For customer, product, and purchase information.
  - ***Accounting***: For payments, receipts, account balances, assets and other accounting information.
  - ***Human resources***: For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.
  - ***Manufacturing***: For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items.
  - ***Online retailers***: For sales data noted above plus online order tracking, generation of recommendation lists, and maintenance of online product evaluations.
- **Banking and Finance**
  - ***Banking***: For customer information, accounts, loans, and banking transactions.
  - ***Credit card transactions***: For purchases on credit cards and generation of monthly statements.
- **Finance**
  - For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.
- **Universities**
  - For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).
- **Airlines**
  - For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.
- **Telecommunication**
  - For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

Purpose of Database Systems:

Database systems arose in response to early methods of computerized management of commercial data. *As an example* of such methods, typical of the 1960s, consider part of a ***university organization*** that, among other data, keeps information about all *instructors*, *students*, *departments*, and *course* offerings. One way to keep the information on a computer is to **store** it in ***operating system files***. *To allow users to manipulate the information, the system has a number of application programs that manipulate the files*, including programs to:

- ✓ **Add** new students, instructors, and courses
- ✓ **Register** students for courses and generate class rosters
- ✓ **Assign** grades to students, **compute** grade point averages (GPA), and generate transcripts

INTRODUCTION TO BASIC CONCEPTS OF DATABASE SYSTEMS Cont.

What is Data?

The *raw facts* are called as data. The word "raw" indicates that they have not been processed.

For example, **89** is the data.

What is information?

The *processed data* is known as information.

For example, Marks: 89; then it becomes information.

What is Knowledge?

1. Knowledge refers to the practical use of information.
2. Knowledge necessarily involves a personal experience.

DATA/INFORMATION PROCESSING:

The process of converting the data (raw facts) into meaningful information is called as data/information processing.

Note: In business, processing knowledge is more useful to make decisions for any organization.

DIFFERENCE BETWEEN DATA AND INFORMATION:

DATA	INFORMATION
a) Raw facts.	a) Processed data
b) It is in unorganized form	b) It is in organized form
c) Data doesn't help in Decision making process	c) Information helps in Decision making process

FILE ORIENTED APPROACH:

The earliest business computer systems were used to process business records and produce information. They were generally faster and more accurate than equivalent manual systems.

These systems stored groups of records in separate files, and so they were called **file processing systems.**

1. File system is a collection of data. Any management with the file system, *user has to write the procedures*.
2. File system gives the details of the data representation and Storage of data.
3. In File, system storing and retrieving of data cannot be done efficiently.
4. Concurrent access to the data in the file system has many problems like a *Reading* the file while other *deleting* some information, *updating* some information.
5. File system does not provide *crash recovery mechanism*.

   For example, while we are entering some data into the file if System crashes then content of the file is *completely lost*.
6. Protecting a file under file system is *very difficult*.

The typical file-oriented system is supported by a conventional operating system. **Permanent records** are stored in *various files* and a *number of different application programs* are written to extract records from and add records to the appropriate files.

DISADVANTAGES OF FILE-ORIENTED SYSTEM:

The following are the disadvantages of File-Oriented System:

1. Data Redundancy and Inconsistency:

Since files and application programs are created by different programmers over a long period, the files are likely to be:

a) Having different *formats* and the programs may be written in *several programming languages*.

b) Moreover, the same piece of information may be duplicated in several places.

⇒ This redundancy leads to:

- Higher ***storage*** and ***access cost***.
- In addition, it may lead to ***data inconsistency***.

2. Difficulty in Accessing Data:

The conventional file processing environments do not allow needed data to be retrieved in a convenient and efficient manner. Better data retrieval system must be developed for general use.

3. **Data Isolation:**

   Since data is scattered in various files, and files may be in different formats, *it is difficult to write new application programs to retrieve the appropriate data*.

4. **Concurrent Access Anomalies:**

   In order to improve the overall performance of the system and obtain a faster response time, many systems allow multiple users to update the data simultaneously. *In such an environment, interaction of concurrent updates may result in inconsistent data*.

5. **Security Problems:**

   Not every user of the database system should be able to access all the data. For example, in banking system, payroll personnel need only that part of the database that has information about various bank employees. *They do not need access to information about customer accounts. It is difficult to enforce such security constraints*.

6. **Integrity Problems:**

   The data values stored in the database must satisfy certain types of consistency constraints. For example, the balance of a bank account may never fall below a prescribed amount. These constraints are enforced in the system by adding appropriate code in the various application programs. *When new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items for different files*.

7. **Atomicity Problem:**

   A computer system like any other mechanical or electrical device is subject to failure. *In many applications, it is crucial to ensure that once a failure has occurred and has been detected, the data are restored to the consistent state existed prior to the failure*.

Example:

Consider part of a savings-bank enterprise that keeps information about all customers and savings accounts. One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including:

- A program to debit or credit an account
- A program to add a new account
- A program to find the balance of an account
- A program to generate monthly statements

Programmers wrote these application programs to meet the needs of the bank. New application programs are added to the system as the need arises. For example, suppose that the savings bank decides to offer checking accounts.

As a result, the bank creates new permanent files that contain information about all the checking accounts maintained in the bank, and it may have to write new application programs to deal with situations that do not arise in savings accounts, such as overdrafts. Thus, as time goes by, the system acquires more files and more application programs.

The system stores permanent records in various files, and it needs different Application programs to extract records from, and add records to, the appropriate files. Before database management systems (DBMS) came along, organizations usually stored information in such systems. ***Organizational information*** *in a file- processing system has a number of major disadvantages*:

1. **Data Redundancy and Inconsistency:**

   The address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost. In, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree. For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.

2. **Difficulty in Accessing Data:**

   Suppose that one of the bank officers needs to find out the names of all customers who live within a particular postal-code area. The officer asks the data-processing department to generate such a list. Because there is no application program to generate that. The bank officer has now two choices: either obtain the list of all customers and extract the needed information manually or ask a system programmer to write the necessary application program. Both alternatives are obviously unsatisfactory.

3. **Data Isolation:**

Because data are scattered in various files and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

4. **Integrity Problems:**

The balance of a bank account may never fall below a prescribed amount (say, $25). Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

5. **Atomicity Problems:**

A computer system, like any other mechanical or electrical device, is subject to failure. In many applications, it is important that, if a **failure** occurs, the data be restored to the consistent state that existed prior to the failure.

Consider a program to transfer **$50** from account ***A*** to account ***B***. If a system failure occurs during the execution of the program, it is possible that the **$50** was removed from account ***A*** but was not credited to account ***B***, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be ***atomic***—**it must happen in its entirety or not at all**. It is difficult to ensure atomicity in a conventional file-processing system.

6. **Concurrent-Access Anomalies:**

For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data.

Consider bank account ***A***, containing **$500**. If two customers withdraw funds (say **$50** and **$100** respectively) from account ***A*** at about the same time, the result of the concurrent executions may leave the account in an **incorrect** (or **inconsistent**) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value **$500**, and write back **$450** and **$400**, respectively. Depending on which one writes the value last, the account may contain $450 or **$400**, rather than the correct value of **$350**. To guard against this possibility, the system must maintain some form of supervision. However, supervision is difficult to provide

because data may be accessed by many different application programs that have not been coordinated previously.

7. **Security Problems:**

Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer accounts. But, since application programs are added to the system in an ad hoc manner, enforcing such security constraints is difficult. These difficulties, among others, prompted the development of database systems.

History of Database Systems:

1950s and early 1960s:

- Magnetic tapes were developed for data storage
- Data processing tasks such as payroll were automated, with data stored on tapes.
- Data could also be input from punched card decks, and output to printers.
- Late 1960s and 1970s: The use of hard disks in the late 1960s changed the scenario for data processing greatly, since hard disks allowed direct access to data.
- With disks, network and hierarchical databases could be created that allowed data structures such as lists and trees to be stored on disk. Programmers could construct and manipulate these data structures.
- With disks, network and hierarchical databases could be created that allowed data structures such as lists and trees to be stored on disk. Programmers could construct and manipulate these data structures.
- In the 1970's the EF CODD defined the **Relational Model.**

In the 1980's:

- Initial commercial relational database systems, such as IBM DB2, Oracle, Ingress, and DEC Rdb, played a major role in advancing techniques for efficient processing of declarative queries.
- In the early 1980s, relational databases had become competitive with network and hierarchical database systems even in the area of performance.
- The 1980s also saw much research on parallel and distributed databases, as well as initial work on object-oriented databases.

Early 1990s:

- The SQL language was designed primarily in the 1990's.
- And this is used for the transaction processing applications.
- Decision support and querying re-emerged as a major application area for databases.
- Database vendors also began to add object-relational support to their databases.

Late 1990s:

- The major event was the explosive growth of the World Wide Web.
- Databases were deployed much more extensively than ever before. Database systems now had to support very high transaction processing rates, as well as very high reliability and 24 * 7 availability (availability 24 hours a day, 7 days a week, meaning no downtime for scheduled maintenance activities).
- Database systems also had to support Web interfaces to data.

The Evolution of Database systems:

The Evolution of Database systems are as follows:

1. File Management System
2. Hierarchical database System
3. Network Database System
4. Relational Database System

1) File Management System:

The file management system also called as FMS in short is one in which all data is stored on a single large file. The main disadvantage in this system is searching a record or data takes a long time. This lead to the introduction of the concept, of indexing in this system. Then also the FMS system had lot of drawbacks to name a few like updating or modifications to the data cannot be handled easily, sorting the records took long time and so on. All these drawbacks led to the introduction of the Hierarchical Database System.

2) Hierarchical Database System:

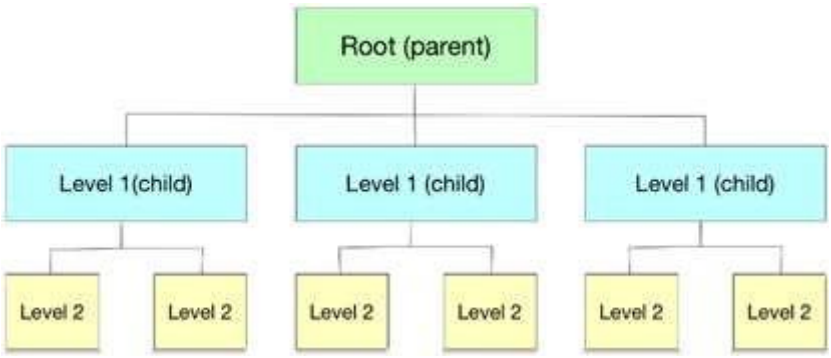


Fig: Hierarchical Database System

The previous system FMS drawback of accessing records and sorting records which took a long

time was removed in this by the introduction of parent-child relationship between records in database. The origin of the data is called the root from which several branches have data at different levels and the last level is called the leaf. The main drawback in this was if there is any modification or addition made to the structure then the whole structure needed alteration, which made the task a tedious one. In order to avoid this next system took its origin, which is called as the Network Database System.

3) Network Database System:

In this, the main concept of many-many relationships got introduced. But this also followed the same technology of pointers to define relationships with a difference in this made in the introduction if grouping of data items as sets.

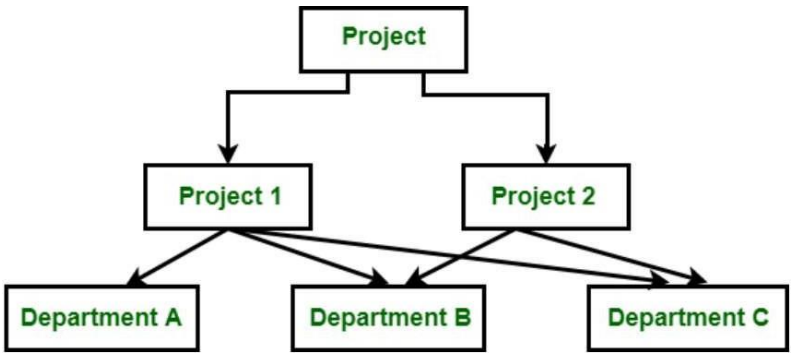


Fig: Network Database System

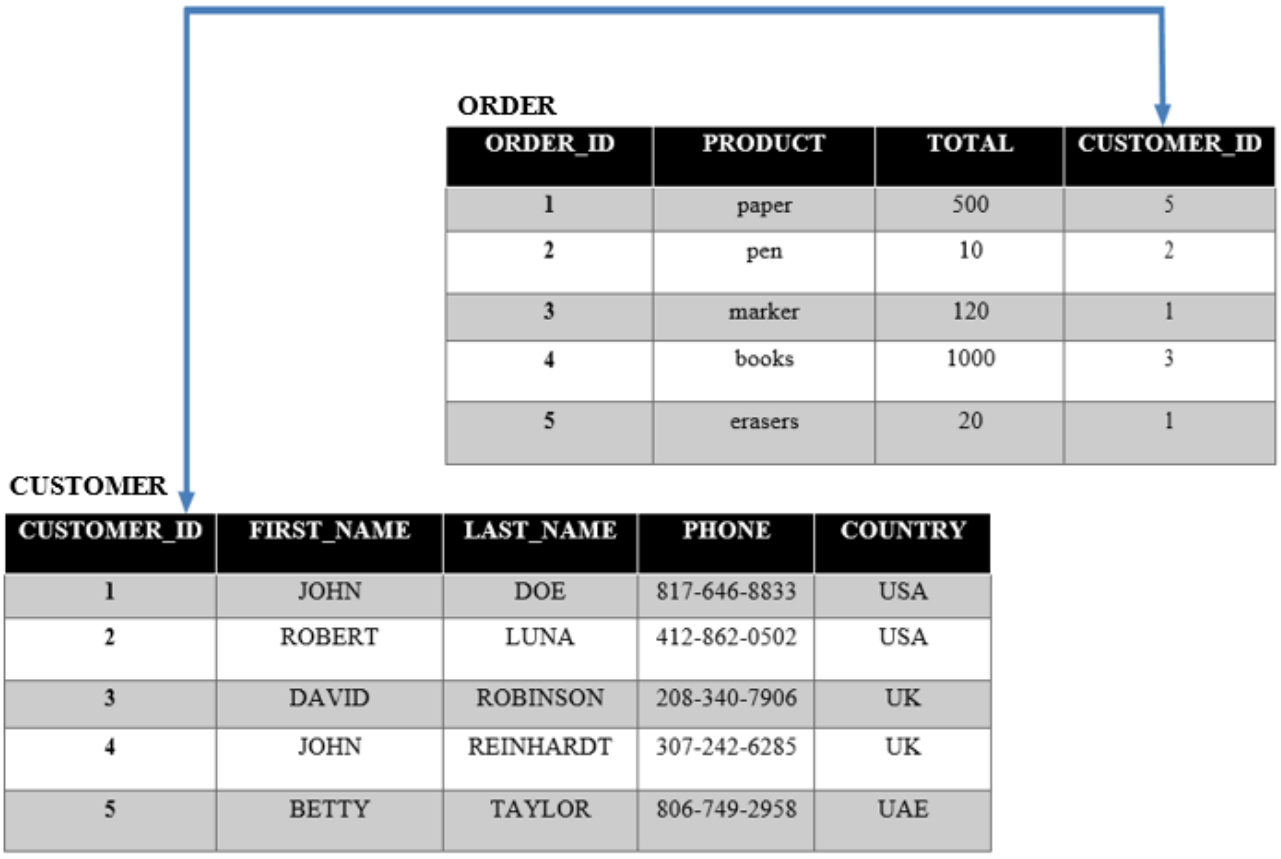
4) Relational Database System:

In order to overcome all the drawbacks of the previous systems, the Relational Database System got introduced in which data get organized as tables and each record forms a row with many fields or attributes in it. Relationships between tables are also formed in this system.

CUSTOMER

CUSTOMER_ID	FIRST_NAME	LAST_NAME	PHONE	COUNTRY
1	JOHN	DOE	817-646-8833	USA
2	ROBERT	LUNA	412-862-0502	USA
3	DAVID	ROBINSON	208-340-7906	UK
4	JOHN	REINHARDT	307-242-6285	UK
5	BETTY	TAYLOR	806-749-2958	UAE

Fig*: Relational Database System (RDBS)*



ORDER

ORDER_ID	PRODUCT	TOTAL	CUSTOMER_ID
1	paper	500	5
2	pen	10	2
3	marker	120	1
4	books	1000	3
5	erasers	20	1

CUSTOMER

CUSTOMER_ID	FIRST_NAME	LAST_NAME	PHONE	COUNTRY
1	JOHN	DOE	817-646-8833	USA
2	ROBERT	LUNA	412-862-0502	USA
3	DAVID	ROBINSON	208-340-7906	UK
4	JOHN	REINHARDT	307-242-6285	UK
5	BETTY	TAYLOR	806-749-2958	UAE

Fig: *Relationship in RDBS*

- **Advantages of DBMS:**

1) **Controlling of Redundancy:** Data redundancy refers to the duplication of data (i.e storing same data multiple times). In a database system, by having a centralized database and centralized control of data by the DBA the unnecessary duplication of data is avoided. It also eliminates the extra time for processing the large volume of data. It results in saving the storage space.

2) **Improved Data Sharing:** DBMS allows a user to share the data in any number of application programs.

3) **Data Integrity:** Integrity means that the data in the database is accurate. Centralized control of the data helps in permitting the administrator to define integrity constraints to the data in the database. For example: in customer database, we can enforce an integrity that it must accept the customer only from Noida and Meerut city.

4) **Security:** Having complete authority over the operational data, enables the DBA in ensuring that the only mean of access to the database is through proper channels. The DBA can define authorization checks to be carried out whenever access to sensitive data is attempted.

5) **Data Consistency:** By eliminating data redundancy, we greatly reduce the opportunities for

inconsistency. For example: is a customer address is stored only once, we cannot have disagreement on the stored values. Also updating data values is greatly simplified when each value is stored in one place only. Finally, we avoid the wasted storage that results from redundant data storage.

6) **Efficient Data Access:** In a database system, the data is managed by the DBMS and all access to the data is through the DBMS providing a key to effective data processing

7) **Enforcements of Standards**: With the centralized of data, DBA can establish and enforce the data standards which may include the naming conventions, data quality standards etc.

8) **Data Independence**: Ina database system, the database management system provides the interface between the application programs and the data. When changes are made to the data representation, the Meta data obtained by the DBMS is changed but the DBMS is continues to provide the data to application program in the previously used way. The DBMs handles the task of transformation of data wherever necessary.

9) **Reduced Application Development and Maintenance Time:** DBMS supports many important functions that are common to many applications, accessing data stored in the DBMS, which facilitates the quick development of application.

- **Disadvantages of DBMS:**

1) It is bit complex. Since it supports multiple functionality to give the user the best, the underlying software has become complex. The designers and developers should have thorough knowledge about the software to get the most out of it.

2) Because of its complexity and functionality, it uses large amount of memory. It also needs large memory to run efficiently.

3) DBMS system works on the centralized system, i.e.; all the users from all over the world access this database. Hence any failure of the DBMS, will impact all the users.

4) DBMS is generalized software, i.e.; it is written work on the entire systems rather specific one. Hence, some of the application will run slow.

View of Data

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an ***abstract*** view of the data. ***That is, the system hides certain details of how the data are stored and maintained***.

Data Abstraction

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:

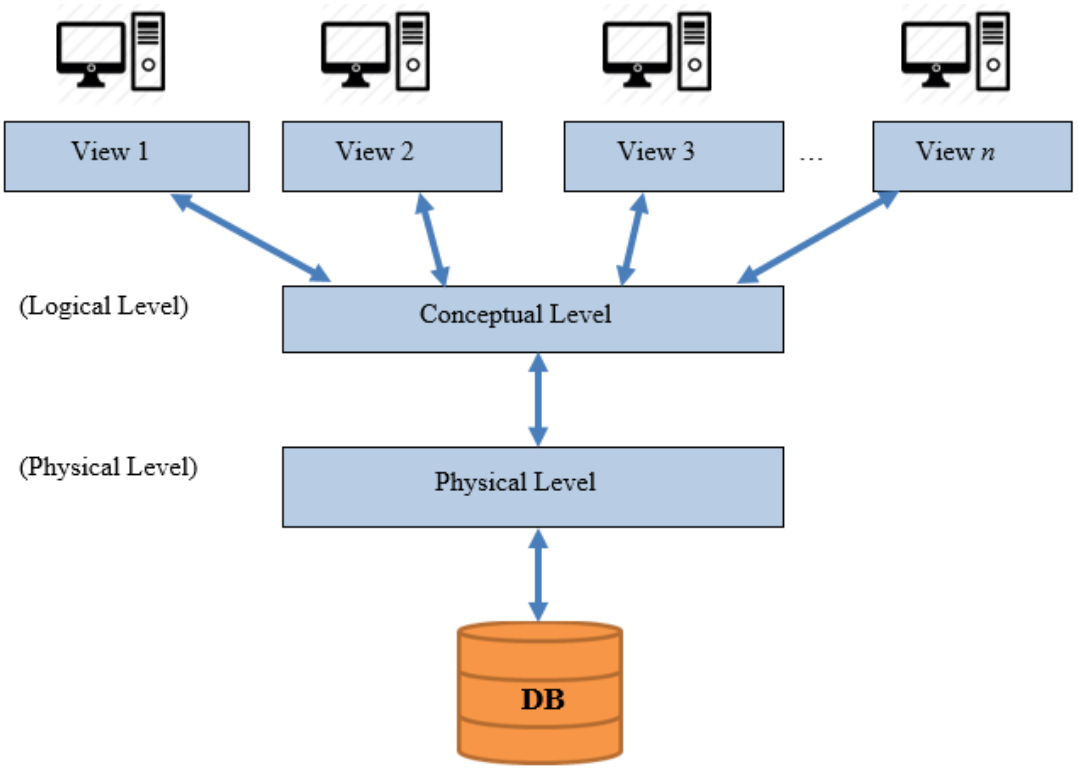


Fig: Levels of Abstraction in a DBMS

A) **Physical level (or Internal View / Schema)**: The lowest level of abstraction describes ***how*** the data are ***actually stored***. The physical level describes complex low-level data structures in detail.

B) **Logical level (or Conceptual View / Schema)**: The next-higher level of abstraction describes ***what*** **data** are stored in the database, and *what* **relationships** exist among those data. The logical level thus describes *the entire database in terms of a small number of relatively simple structures*.

Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as **physical data independence**.

Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

C) **View level (or External View / Schema):** The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database. Figure above shows the relationship among the three levels of abstraction.

An analogy to the concept of **data types** in **programming languages** may clarify the distinction among levels of abstraction. Many high-level programming languages support the notion of a **structured type**. For example, we may describe a record as follows:

```
type
  Instructor = record
  ID : char (5);
  name : char (20);
  dept name : char (20);
  salary : numeric (8,2);
end;
```

This code defines a new record type called ***instructor*** with **four fields**. Each field has a ***name*** and a *type* associated with it. A university organization may have several such record types, including

- ***Department***, with fields *dept_name*, *building*, and *budget*
- ***Course***, with fields *course_id*, *title*, *dept_name*, and *credits*
- ***Student***, with fields *ID*, *name*, *dept_name*, and *tot_cred*

At the physical level, an ***Instructor***, ***Department***, or ***Student*** record can be described as a block of consecutive storage locations. The **compiler hides this level of detail from programmers**. Similarly, the **database system hides many of the lowest-level storage details from database**

programmers. Database administrators, on the other hand, may be aware of certain details of the physical organization of the data.

At the logical level, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. Programmers using a programming language work at this level of abstraction. Similarly, database administrators usually work at this level of abstraction.

Finally, at the view level, computer users see a **set of application programs** that **hide details of the data types.** At the view level, several views of the database are defined, and a database user sees some or all of these views.

In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing certain parts of the database.

For example, clerks in the university registrar office can see only that part of the database that has information about students; they cannot access information about salaries of instructors.

Instances and Schemas

Databases change over time as information is inserted and/or deleted. T*he collection of information stored in the database at a particular moment is called an* **instance** of the database. *The overall design of the database is called the* **database schema**. Schemas are changed infrequently, if at all. The concept of database schemas and instances can be understood by analogy to a **program** written in a programming language. A database schema corresponds to the **variable** declarations (along with associated **type definitions**) in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an *instance* of a database schema.

Database systems have several schemas, partitioned according to the levels of abstraction. The **physical schema** *describes the database design at the physical level*, while the **logical schema** *describes the database design at the logical level*. A database may also have several schemas at the view level, sometimes called **subschemas**, which describe different views of the database. Of these, the logical schema is by far the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema. The physical schema is hidden beneath the logical

schema, and can usually be changed easily without affecting application programs. Application programs are said to exhibit **physical data independence** if they do not depend on the physical schema, and thus need not be rewritten if the physical schema changes.

Data Models

Underlying the structure of a database is the **data model**: *a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. A data model provides a way to describe the design of a database at the physical, logical, and view levels.*

The data models can be classified into four different categories:

A). **Relational Model**. The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as **relations**. The relational model is an example of a record-based model.
Record-based models are so named because the database is structured in fixed-format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record type. The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model.

B). **Entity-Relationship Model**. The entity-relationship (E-R) data model uses a collection of basic objects, called *entities*, and *relationships* among these objects.
An entity is a "thing" or "object" in the real world that is distinguishable from other objects. The entity-relationship model is widely used in database design.

C). **Object-Based Data Model**. Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology. This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity. The object-relational data model combines features of the object-oriented data model and relational data model.

D). **Semi-structured Data Model**. The semi-structured data model permits the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast to the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The **Extensible Markup Language (XML)** is widely used to represent semi-structured data.

- Historically, the **network data model** and the **hierarchical data model** preceded the relational data model. These models were tied closely to the underlying implementation, and complicated the task of modeling data. As a result they are used little now, except in old database code that is still in service in some places.

Database Languages

A database system provides a **data-definition language** to specify the database schema and a **data-manipulation language** to express database queries and updates. In practice, the data- definition and data-manipulation languages are **not two separate languages**; instead they simply form parts of a single database language, such as the widely used SQL language.

(1) Data-Manipulation Language

A **data-manipulation language (DML)** is a language that enables users to access or manipulate data as organized by the appropriate data model. The types of access are:

- ***Retrieval*** of information stored in the database
- ***Insertion*** of new information into the database
- ***Deletion*** of information from the database
- ***Modification*** of information stored in the database

There are basically two types of DML:

(a) Procedural DMLs require a user to specify ***what*** data are needed and ***how*** to get those data.

(b) Declarative DMLs (also referred to as **nonprocedural DMLs**) require a user to specify *what* data are needed *without* specifying how to get those data.

Declarative DMLs are usually easier to learn and use than are procedural DMLs. However, since a user does not have to specify how to get the data, the database system has to figure out an efficient means of accessing data. A **query** is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a **query language**. Although technically incorrect, it is common practice to use the terms *query language* and *data-manipulation language* synonymously.

(2) Data-Definition Language (DDL)

We specify a database schema by a set of definitions expressed by a special language called a **Data-Definition Language** (**DDL**). The DDL is also used to specify additional properties of the data.

We specify the ***storage structure*** and ***access methods*** used by the database system by a set of statements in a special type of DDL called a **data storage and definition** language. These statements define the implementation details of the database schemas, *which are usually hidden from the users*. The data values stored in the database must satisfy certain ***consistency constraints***.

For example, suppose the university requires that the ***account balance*** of a department must never be **negative**. The DDL provides facilities to specify such constraints. The database system checks these constraints *every time the database is updated*. In general, a constraint can be an arbitrary predicate pertaining to the database. However, arbitrary predicates may be costly to test. Thus, database systems implement integrity constraints that can be tested with minimal overhead.

A) **Domain Constraints**. *A domain of possible values must be associated with every attribute* (for example, integer types, character types, date/time types). Declaring an attribute to be of a particular domain acts as a constraint on the values that it can take. Domain constraints are the most elementary form of integrity constraint. They are tested easily by the system whenever a new data item is entered into the database.

B) **Referential Integrity**. There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation (referential integrity). For example, the department listed for each course must be one that actually exists. More precisely, the *dept name* value in a *course* record must appear in the *dept name* attribute of some record of the *department* relation.
Database modifications can cause violations of referential integrity. When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.

C) **Assertions**. *An assertion is any condition that the database must always satisfy*. Domain constraints and referential-integrity constraints are special forms of assertions. However, there are many constraints that we cannot express by using only these special forms. For example, "Every department must have at least five courses offered every semester" must be expressed as an assertion. When an

assertion is created, the system tests it for validity. If the assertion is valid, then any future modification to the database is allowed only if it does not cause that assertion to be violated.

D) **Authorization**. We may want to differentiate among the users as far as the type of access they are permitted on various data values in the database. These differentiations are expressed in terms of **authorization**, the most common being: **read authorization**, which allows reading, but not modification, of data; **insert authorization**, which allows insertion of new data, but not modification of existing data; **update authorization**, which allows modification, but not deletion, of data; and **delete authorization**, which allows deletion of data. We may assign the user all, none, or a combination of these types ofauthorization.

The DDL, just like any other programming language, gets as input some instructions (statements) and generates some output. The output of the DDL is placed in the **data dictionary**, which contains **metadata**—that is, *data about data*. The data dictionary is considered to be a ***special type of table that can only be accessed and updated by the database system itself (not a regular user)***. The database system consults the data dictionary before reading or modifying actual data.

Lec 5

Database Languages

A database system provides a **data-definition language** to specify the database schema and a **data-manipulation language** to express database queries and updates. In practice, the data- definition and data-manipulation languages are not two separate languages; instead they simply form parts of a single database language, such as the widely used SQL language.

(1) Data-Manipulation Language

A **data-manipulation language (DML)** is a language that enables users to access or manipulate data as organized by the appropriate data model. The types of access are:

- ***Retrieval*** of information stored in the database
- ***Insertion*** of new information into the database
- ***Deletion*** of information from the database
- ***Modification*** of information stored in the database

A **query** is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a **query language**.

(2) Data-Definition Language (DDL)

We specify a database schema by a set of definitions expressed by a special language called a **data-definition language** (**DDL**). The DDL is also used to specify additional properties of the data.

We specify the ***storage structure*** and ***access methods*** used by the database system by a set of statements in a special type of DDL called a **data storage and definition** language. These statements define the implementation details of the database schemas, *which are usually hidden from the users*.

The data values stored in the database must satisfy certain ***consistency constraints***.

A) **Domain Constraints**. A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types). Declaring an attribute to be of a particular domain acts as a constraint on the values that it can take. Domain constraints are the most elementary form of integrity constraint. They are tested easily by the system whenever a new data item is entered into the database.

B) **Referential Integrity**. There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation (referential integrity). For example, the department listed for each course must be one that actually exists. More precisely, the *dept name* value in a *course* record must appear in the *dept name* attribute of some record of the *department* relation.

Database modifications can cause violations of referential integrity. When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.

C) **Assertions**. An assertion is any condition that the database must always satisfy. Domain constraints and referential-integrity constraints are special forms of assertions. However, there are many constraints that we cannot

express by using only these special forms. For example, “Every department must have at least five courses offered every semester” must be expressed as an assertion. When an assertion is created, the system tests it for validity. If the assertion is valid, then any future modification to the database is allowed only if it does not cause that assertion to be violated.

D) **Authorization**. We may want to differentiate among the users as far as the type of access they are permitted on various data values in the database. These differentiations are expressed in terms of **authorization**, the most common being: **read authorization**, which allows reading, but not modification, of data; **insert authorization**, which allows insertion of new data, but not modification of existing data; **update authorization**, which allows modification, but not deletion, of data; and **delete authorization**, which allows deletion of data. We may assign the user all, none, or a combination of these types of authorization.

Data Dictionary

We can define a data dictionary as a DBMS component that stores the definition of data characteristics and relationships. You may recall that such “data about data” were labeled metadata. The DBMS data dictionary provides the DBMS with its self-describing characteristic.

The two main types of data dictionary exist, integrated and stand alone. An integrated data dictionary is included with the DBMS. For example, all relational DBMSs include a built in data dictionary or system catalog that is frequently accessed and updated by the RDBMS. Other DBMSs especially older types, do not have a built in data dictionary instead the DBA may use

third party stand-alone data dictionary systems.

Data dictionaries can also be classified as active or passive. An active data dictionary is automatically updated by the DBMS with every database access, thereby keeping its access information up-to-date. A passive data dictionary is not updated automatically and usually requires a batch process to be run. Data dictionary access information is normally used by the DBMS for query optimization purpose.

The data dictionary's main function is to store the description of all objects that interact with the database. Integrated data dictionaries tend to limit their metadata to the data managed by the DBMS. Stand-alone data dictionary systems are more usually more flexible and allow the DBA to describe and manage all the organization's data, whether or not they are computerized. Whatever the data dictionary's format, its existence provides database designers and end users with a much improved ability to communicate. In addition, the data dictionary is the tool that helps the DBA to resolve data conflicts.

Database Administrators

Database Administrator (DBA): is a person/team who defines the schema and also controls the 3 levels of database. The DBA will then create a new account id and password for the user if he/she needs to access the database.

- DBA is also responsible for providing security to the database and he allows only the authorized users to access/modify the database.
- DBA also monitors the recovery and backup and provide technical

support.

- The DBA has a DBA account in the DBMS which called a system or super user account.
- DBA repairs damage caused due to hardware and/or software failures.

Database Users and User Interfaces

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

1- **Naive users / End Users** are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. For example, a bank teller who needs to transfer $50 from account *A* to account *B* invokes a program called *transfer*. This program asks the teller for the amount of money to be transferred, the account from which the money is to be transferred, and the account to which the money is to be transferred.

2- **Application programmers** are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. **Rapid application development (RAD)** tools are tools that enable an application programmer to construct forms and reports without writing a program. There are also special types of programming languages that combine imperative control structures (for example, for loops, while loops and if-then-else statements) with statements of the data manipulation language. These languages, sometimes called *fourth-generation languages*, often include special features to

facilitate the generation of forms and the display of data on the screen. Most major commercial database systems include a fourth generation language.

3- **Sophisticated users** interact with the system without writing programs. Instead, they form their requests in a database query language. They submit each such query to a **query processor**, whose function is to break down DML statements into instructions that the storage manager understands. Analysts who submit queries to explore data in the database fall in this category. They use the tools to perform their task such as:

   a- **Online analytical processing (OLAP)** tools simplify analysts' tasks by letting them view summaries of data in different ways. For instance, an analyst can see total sales by region (for example, North, South, East, and West), or by product, or by a combination of region and product (that is, total sales of each product in each region). The tools also permit the analyst to select specific regions, look at data in more detail (for example, sales by city within a region) or look at the data in less detail (for example, aggregate products together by category).

   b- Another class of tools for analysts is **data mining** tools, which help them, find certain kinds of patterns in data.

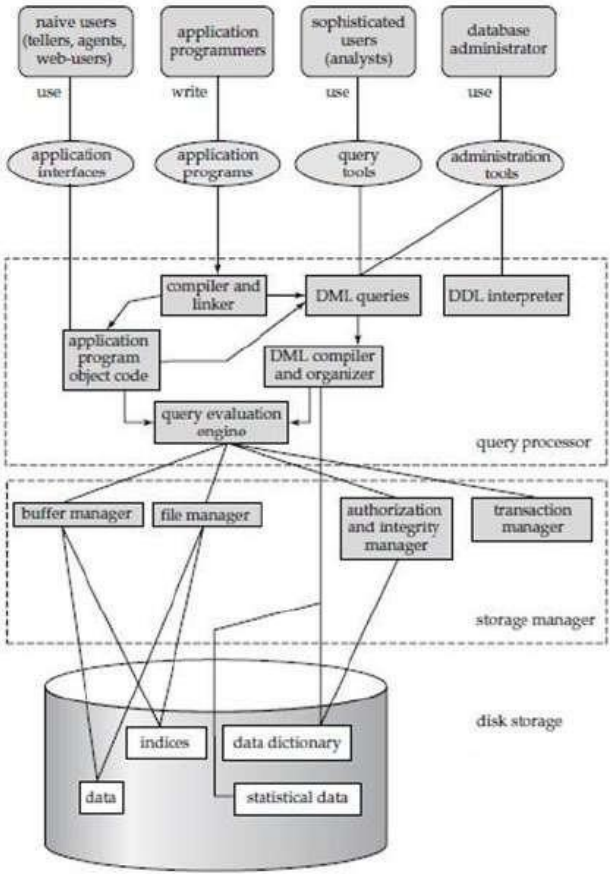
4- **Specialized users** are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.

Among these applications are computer-aided design systems, knowledge base and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

Lec 6

Database Architecture:

The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs. Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines. Database systems can also be designed to exploit parallel computer architectures. Distributed databases span multiple geographically separated machines.



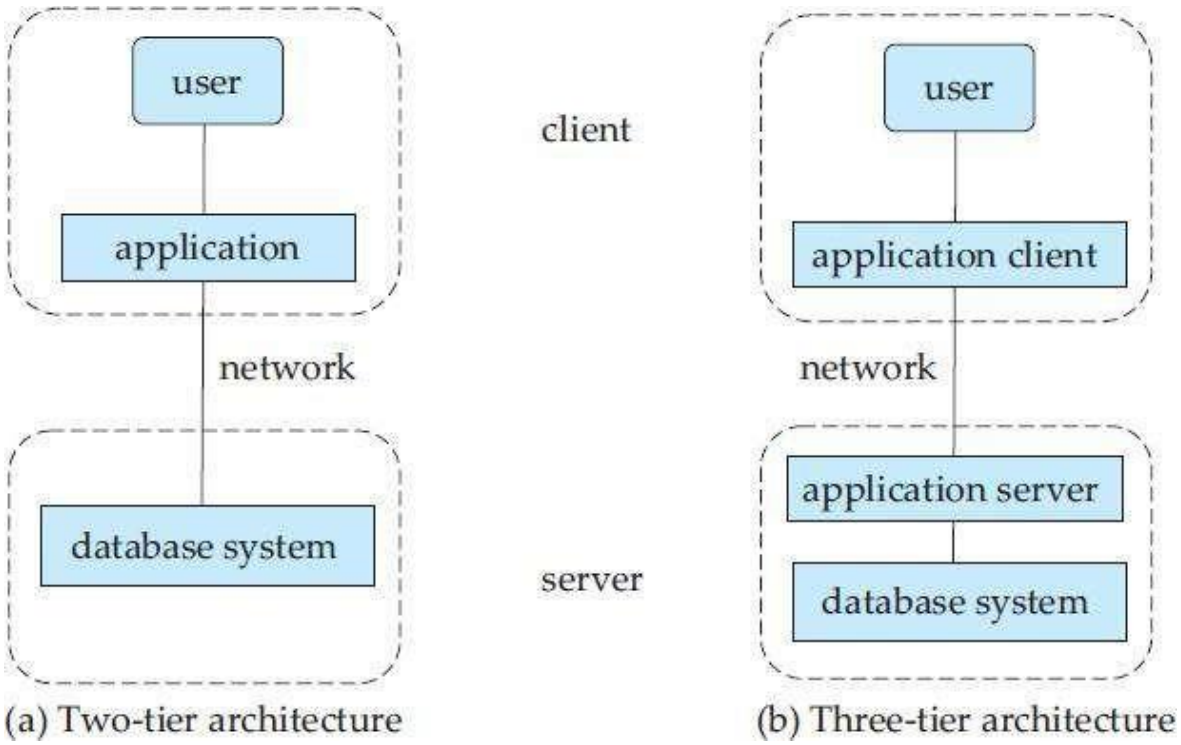
Database System Architecture

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage **manager and** the **query processor** components. The storage manager is important because databases typically require a large amount of storage space. The query processor is important because it helps the database system simplify and facilitate access to data.

It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level.

Database applications are usually partitioned into two or three parts, as in Figure 1.4. In two-tier architecture, the application resides at the client machine, where it invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server. In contrast, in a three-tier architecture, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an application server, usually through a forms interface.

The application server in turn communicates with a database system to access data. The business logic of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients. Three-tier applications are more appropriate for large applications, and for applications that run on the WorldWideWeb.



Two-tier and three-tier architectures.

Query Processor:

The query processor components include

DDL interpreter, which interprets DDL statements and records the definitions in the data dictionary.

DML compiler, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.

A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs **query optimization**, that is, it picks the lowest cost evaluation plan from among the alternatives.

Query evaluation engine, which executes low-level instructions generated by the DML compiler.

Storage Manager:

A *storage manager* is a program module that provides the interface between the lowlevel data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager. The raw data are stored on the disk using the file system, which is usually provided by a conventional operating system. The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for storing, retrieving, and updating data in the database.

The storage manager components include:

Authorization and integrity manager, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.

Transaction manager, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.

File manager, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

Buffer manager, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

Transaction Manager:

A **transaction** is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any database-consistency constraints. That is, if the database was consistent when a transaction started, the database must be consistent when the transaction successfully terminates. **Transaction - manager** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.

Conceptual Database Design - Entity Relationship(ER) Modeling:

Database Design Techniques

1. ER Modeling (Top down Approach)
2. Normalization (Bottom Up approach)

What is ER Modeling?

A graphical technique for understanding and organizing the data independent of the actual database implementation

We need to be familiar with the following terms to go further.

Entity

Any thing that has an independent existence and about which we collect data. It is also known as entity type. In ER modeling, notation for entity is given below.

Entity instance

Entity instance is a particular member of the entity type. Example for entity instance : A particular employee **Regular Entity**

An entity which has its own key attribute is a regular entity. Example for regular entity : Employee.

Weak entity

An entity which depends on other entity for its existence and doesn't have any key attribute of its own is a weak entity.

Example for a weak entity: In a parent/child relationship, a parent is considered as a strong entity and the child is a weak entity.

In ER modeling, notation for weak entity is given below.

Attributes

Properties/characteristics which describe entities are called attributes. In ER modeling, notation for attribute is given below.

Domain of Attributes

The set of possible values that an attribute can take is called the domain of the attribute. For example, the attribute day may take any value from the set {Monday, Tuesday ... Friday}. Hence this set can be termed as the domain of the attribute day.

Key attribute

The attribute (or combination of attributes) which is unique for every entity instance is called key attribute.

E.g the employee_id of an employee, pan_card_number of a person etc.If the key attribute consists of two or more attributes in combination, it is called a composite key.

In ER modeling, notation for key attribute is given below.

Simple attribute

If an attribute cannot be divided into simpler components, it is a simple attribute. Example for simple attribute : employee_id of an employee.

Composite attribute

If an attribute can be split into components, it is called a composite attribute.

Example for composite attribute : Name of the employee which can be split into First_name, Middle_name, and Last_name.

Single valued Attributes

If an attribute can take only a single value for each entity instance, it is a single valued attribute. example for single valued attribute : age of a student. It can take only one value for a particular student.

Multi-valued Attributes

If an attribute can take more than one value for each entity instance, it is a multi-valued attribute. Multi-valued

example for multi valued attribute : telephone number of an employee, a particular employee may have multiple telephone numbers.

In ER modeling, notation for multi-valued attribute is given below.

Stored Attribute

An attribute which need to be stored permanently is a stored attribute Example for stored attribute : name of a student

Derived Attribute

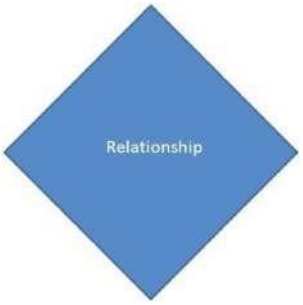
An attribute which can be calculated or derived based on other attributes is a derived attribute.

Example for derived attribute : age of employee which can be calculated from date of birth and current date. In ER modeling, notation for derived attribute is given below.

Relationships

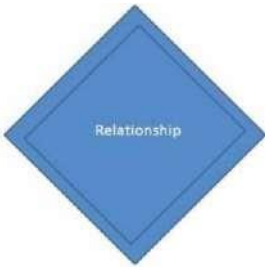
Associations between entities are called relationships

Example : An employee works for an organization. Here "works for" is a relation between the entities employee and organization.



In ER modeling, notation for relationship is given below.

However in ER Modeling, To connect a weak Entity with others, you should use a weak relationship notation as given below



Degree of a Relationship

Degree of a relationship is the number of entity types involved. The n-ary relationship is the general form for degree n. Special cases are unary, binary, and ternary ,where the degree is 1, 2, and 3, respectively.

Example for unary relationship : An employee ia a manager of another employee Example for binary relationship : An employee works-for department.

Example for ternary relationship : customer purchase item from a shop keeper

Cardinality of a Relationship

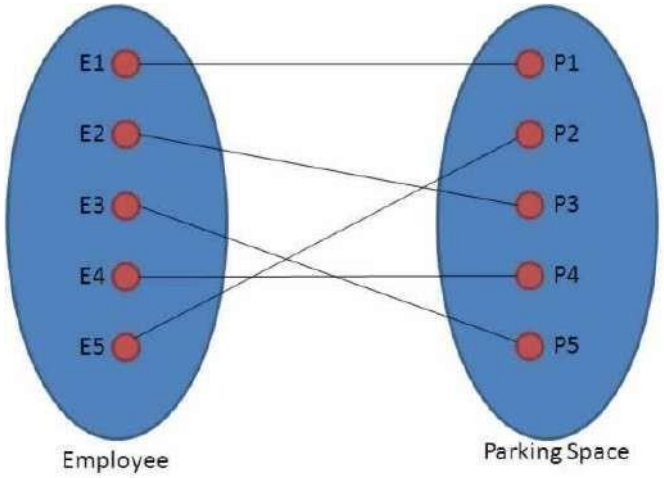
Relationship cardinalities specify how many of each entity type is allowed. Relationships can have four possible connectivities as given below.

1. One to one (1:1) relationship
2. One to many (1:N) relationship
3. Many to one (M:1) relationship

4. Many to many (M:N) relationship

The minimum and maximum values of this connectivity is called the cardinality of the relationship

Example for Cardinality – One-to-One (1:1)

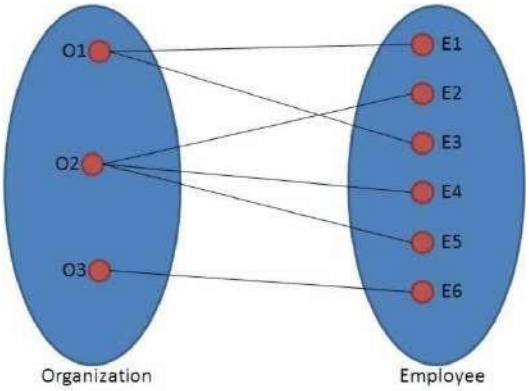


Employee is assigned with a parking space.

One employee is assigned with only one parking space and one parking space is assigned to only one employee. Hence it is a 1:1 relationship and cardinality is One-To-One (1:1)

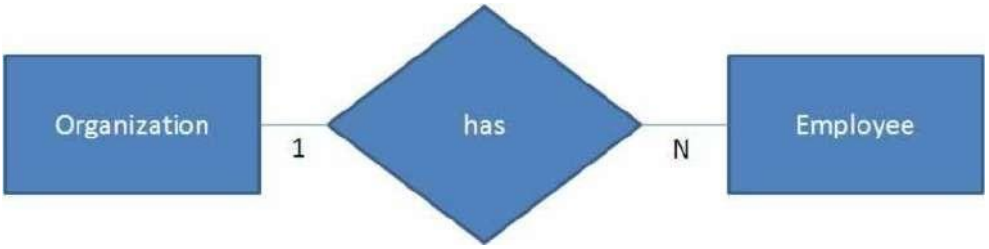
In ER modeling, this can be mentioned using notations as given below

Example for Cardinality – One-to-Many (1:N)



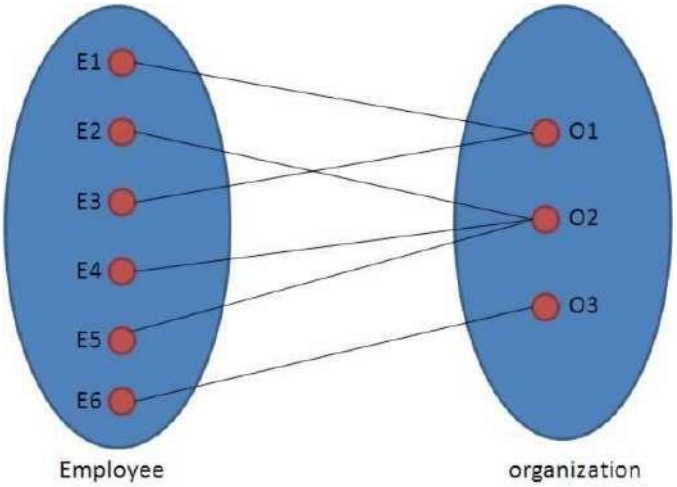
Organization has employees

One organization can have many employees , but one employee works in only one organization. Hence it is a 1:N relationship and cardinality is One-To-Many (1:N)



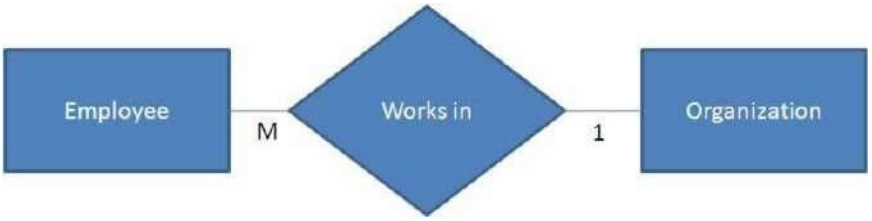
In ER modeling, this can be mentioned using notations as given below

Example for Cardinality – Many-to-One (M :1)



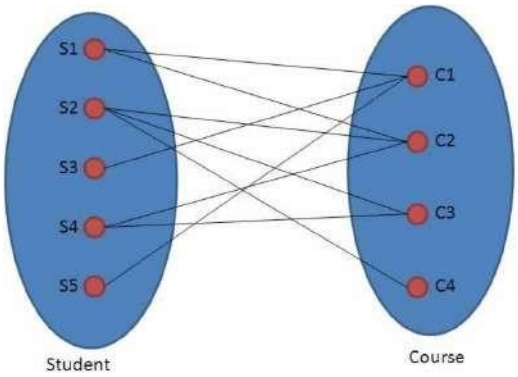
It is the reverse of the One to Many relationship. employee works in organization

One employee works in only one organization But one organization can have many employees. Hence it is a M:1 relationship and cardinality is Many-to-One (M :1)



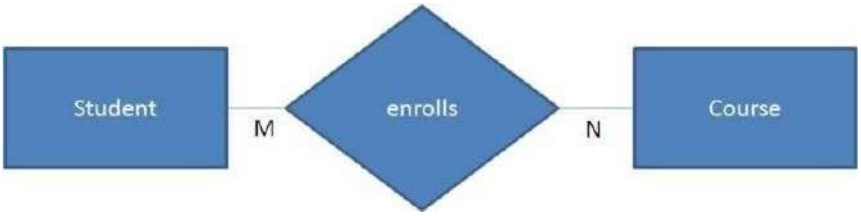
In ER modeling, this can be mentioned using notations as given below.

Cardinality – Many-to-Many (M:N)



Students enrolls for courses

One student can enroll for many courses and one course can be enrolled by many students. Hence it is a M:N relationship and cardinality is Many-to-Many (M:N)



In ER modeling, this can be mentioned using notations as given below

Relationship Participation

1. Total

In total participation, every entity instance will be connected through the relationship to another instance of the other participating entity types

2. Partial

Example for relationship participation

Consider the relationship - Employee is head of the department.

Here all employees will not be the head of the department. Only one employee will be the head of the department. In other words, only few instances of employee entity participate in the above relationship. So employee entity's participation is partial in the said relationship.

However each department will be headed by some employee. So department entity's participation is total in the said relationship.

Advantages and Disadvantages of ER Modeling

Advantages

1. ER Modeling is simple and easily understandable. It is represented in business users language and it can be understood by non-technical specialist.
2. Intuitive and helps in Physical Database creation.
3. Can be generalized and specialized based on needs.

4. Can help in database design.

5. Gives a higher level description of the system.

Disadvantages

1. Physical design derived from E-R Model may have some amount of ambiguities or inconsistency.

2. Sometime diagrams may lead to misinterpretations

Lec 7

Relational Model

Structure of Relational Databases:

A relational database consists of a collection of **tables**, each of which is assigned a unique name. For example, consider the *instructor* table of Figure 1, 2, and 3.

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Figure 1: The *instructor* relation

In general, a row in a table represents a *relationship* among a set of values. Since a table is a collection of such relationships, there is a close correspondence between the concept of *table* and the mathematical concept of *relation*, from which the relational data model takes its name. In mathematical terminology, a *tuple* is simply a sequence (or list) of values. A relationship between *n* values is represented mathematically by an *n-tuple* of values, i.e., a tuple with *n* values, which corresponds to a row in a table.

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

Figure 2: The *course* relation

course_id	*prereq_id*
BIO-301	BIO-101
BIO-399	BIO-101
CS-190	CS-101
CS-315	CS-101
CS-319	CS-101
CS-347	CS-101
EE-181	PHY-101

Figure 3: The *prereq* relation.

Thus, in the relational model the term **relation** is used to refer to a table, while the term **tuple** is used to refer to a row. Similarly, the term **attribute** refers to a column of a table.

Examining Figure 1, we can see that the relation *instructor* has four attributes:

***ID, name, dept name,* and *salary*.**

We use the term **relation instance** to refer to a specific instance of a relation, i.e., containing a specific set of rows. The instance of *instructor* shown in Figure 1 has 12 tuples, corresponding to 12 instructors.

The order in which tuples appear in a relation is irrelevant, since a relation is a *set* of tuples. Thus, whether the tuples of a relation are listed in sorted order, as in Figure 1, or are unsorted, as in Figure 4, does not matter; the relations in the two figures are the same, since both contain the same set of tuples. For ease of exposition, we will mostly show the relations sorted by their first attribute. For each attribute of a relation, there is a set of permitted values, called the **domain** of that attribute. Thus, the domain of the *salary* attribute of the *instructor* relation is the set of all possible salary values, while the domain of the *name* attribute is the set of all possible instructor names.

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Figure 4: Unsorted display of the *instructor* relation.

The **null** value is a special value that signifies that the value is unknown or does not exist. For example, suppose as before that we include the attribute *phone number* in the *instructor* relation. It may be that an instructor does not have a phone number at all, or that the telephone number is unlisted.

Database Schema

When we talk about a database, we must differentiate between the **database schema**, which is the logical design of the database, and the **database instance**, which is a snapshot of the data in the database at a given instant in time.

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Figure 5: The *department* relation.

Consider the *department* relation of Figure 5. The schema for that relation is

***department* (*dept name*, *building*, *budget*)**

Note that the attribute *dept name* appears in both the *instructor* schema and the *department* schema. Using common attributes in relation schemas is one way of relating tuples of distinct relations.

A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation. For example, the ID attribute of the relation instructor is sufficient to distinguish one instructor tuple from another. Thus, ID is a superkey. The name attribute of instructor, on the other hand, is not a superkey, because several instructors might have the same name.

It is possible that several distinct sets of attributes could serve as a candidate key. Suppose that a combination of *name* and *dept name* is sufficient to distinguish among members of the *instructor* relation. Then, both *{ID}* and *{name, dept name}* are candidate keys. Although the attributes *ID* and *name* together can distinguish *instructor* tuples, their combination, *{ID, name}*, does not form a candidate key, since the attribute *ID* alone is a candidate key.

We shall use the term **primary key** to denote a candidate key that is chosen by the database designer as the principal means of identifying tuples within a relation.

It is customary to list the primary key attributes of a relation schema before the other attributes; for example, the *dept name* attribute of *department* is listed first, since it is the primary key. Primary key attributes are also underlined. A relation, say $r1$, may include among its attributes the primary key of another relation, say $r2$. This attribute is called a **foreign key** from $r1$, referencing $r2$.

Schema Diagrams

A database schema, along with primary key and foreign key dependencies, can be depicted by **schema diagrams**. Figure 6 shows the schema diagram for our university organization. Each relation appears as a box, with the relation name at the top in blue, and the attributes listed inside the box. Primary key attributes are shown underlined. Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referenced relation.

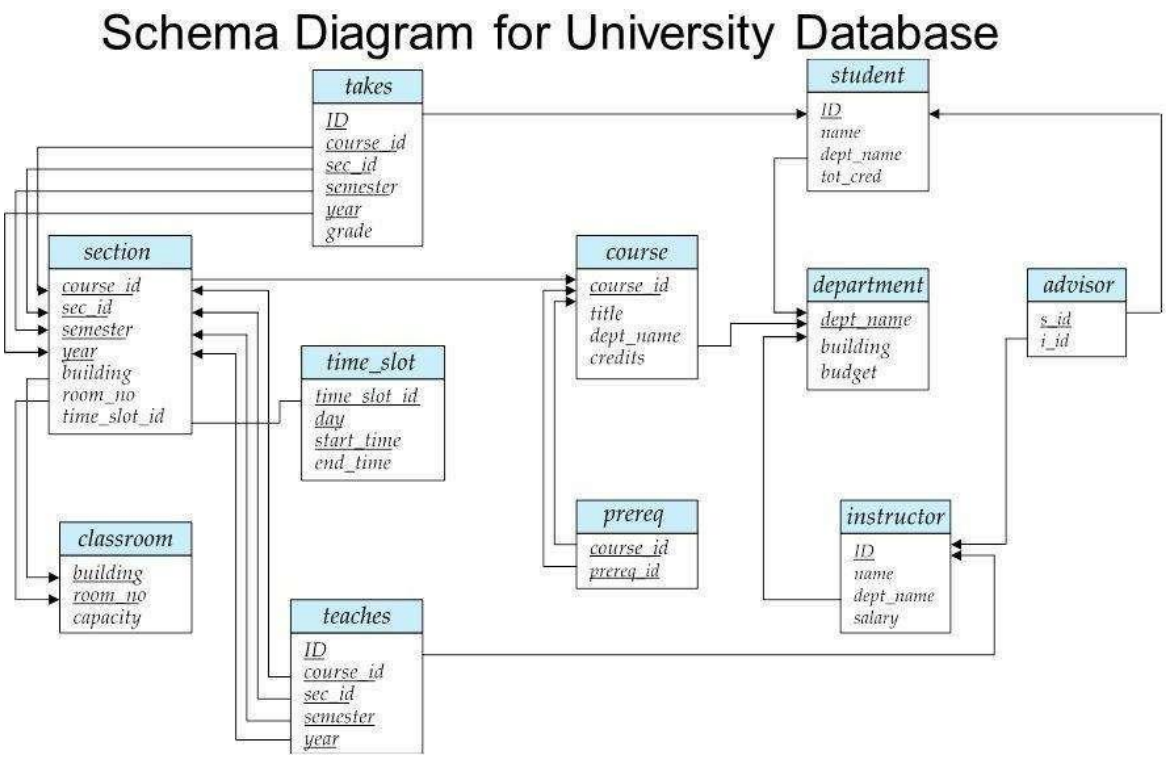


Figure 6 : Schema diagram for the university database.

Structured Query Language (SQL)

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

The table above contains five records (one for each customer) and seven columns (CustomerID, CustomerName, ContactName, Address, City, PostalCode, and Country).

SQL Statements

Most of the actions you need to perform on a database are done with SQL statements.

The following SQL statement selects all the records in the "Customers" table:

Example

```
SELECT * FROM Customers;
```

Keep in Mind That...

- SQL keywords are NOT case sensitive: select is the same as SELECT
- Semicolon after SQL Statements, some database systems require a semicolon at the end of each SQL statement. Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

Some of the Most Important SQL Commands

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database
- CREATE DATABASE - creates a new database
- ALTER DATABASE - modifies a database
- CREATE TABLE - creates a new table
- ALTER TABLE - modifies a table
- DROP TABLE - deletes a table
- CREATE INDEX - creates an index (search key)
- DROP INDEX - deletes an index

SQL SELECT Statement

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

SELECT Syntax

```
SELECT column1, column2, ...
FROM table_name;
```

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

Example

```
SELECT CustomerName, City FROM Customers;
SELECT * FROM Customers;
```

Structured Query Language (SQL)

The SQL WHERE Clause

The WHERE clause is used to filter records.
It is used to extract only those records that fulfill a specified condition.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note: The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

Example
The following SQL statement selects all the customers from the country "Mexico", in the "Customers" table:

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

However, numeric fields should not be enclosed in quotes:
Example

```
SELECT * FROM Customers
WHERE CustomerID=1;
```

The SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Example
The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" column:

```
SELECT * FROM Customers
ORDER BY Country;
```

DESC Example
The following SQL statement selects all customers from the "Customers" table, sorted DESCENDING by the "Country" column:

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

Several Columns Example
The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

The SQL INSERT INTO Statement

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

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

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

Example

The following SQL statement inserts a new record in the "Customers" table:

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

```
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

The SQL UPDATE Statement

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

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

Example

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

The following SQL statement will update the ContactName to "Juan" for all records where country is "Mexico":

Example

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

The SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

```
DELETE FROM table_name WHERE condition;
```

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

Example

The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

```
DELETE FROM table_name;
```

Example

```
DELETE FROM Customers;
```

The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

MIN() Example

The following SQL statement finds the price of the cheapest product:

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

MAX() Example

The following SQL statement finds the price of the most expensive product:

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

The SQL COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criterion.

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

The AVG() function returns the average value of a numeric column.

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

The SUM() function returns the total sum of a numeric column.

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

COUNT() Example

The following SQL statement finds the number of products:

```
SELECT COUNT(ProductID)
FROM Products;
```

AVG() Example

The following SQL statement finds the average price of all products:

```
SELECT AVG(Price)
FROM Products;
```

Note: NULL values are ignored.

SUM() Example

The following SQL statement finds the sum of the "Quantity" fields in the "OrderDetails" table:

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

Note: NULL values are ignored.