Lecture 1

Contents: Introduction; Benefits of database approach; Data abstraction; Instances and schemas of database; Data models; People around DBMS; When DBMS is not useful; Database management software; Exercises

1.1 Introduction

A database management system (DBMS) is a software that manages a large volume of data. It contains interrelated data and a set of programs. It helps users in storing and retrieving information conveniently and efficiently.

Due to technological advancements, nowadays a system can capture a large amount of data. The value of data is widely recognized. Tools are required to update data, manage data and extract information within a short span of time. Such tools are commonly called as database management systems.

A database is a collection of related data. We take here an example of a database that follows relational model. A relation is like a table having different column headings. We shall discuss more on relational model later.

(i) Student file

rollNo	name
101	Pravin Dessai
105	Kavita Nanda
129	Nandita Dalvi
140	Reboti Sinha
150	Rakesh Paroyal

Marks file

rollNo	mark1	mark2	mark3
101	44	62	61
105	75	92	82
129	96	90	97
140	90	63	72
150	66	69	75

Note that the records in *Student* file and *Marks* file are related through field *rollNo*. The given collection of (data) files is a database.

(ii) Student file

rollNo	name
101	Pravin Dessai
105	Kavita Nanda
129	Nandita Dalvi
140	Reboti Sinha
150	Rakesh Paroyal

Product file

pcode	cost	pName
CC231	1421	y-small
ET890	560	x-small
GI501	69	x-medium
KJ885	1129	y-medium
VB907	672	x-big

Records in *Student* file and *Product* file are not related. So, this collection of (data) files is not a database. Rather, it is a collection of two databases, each database containing one file.

We present a simplified view of a database system in Fig. 1.1.

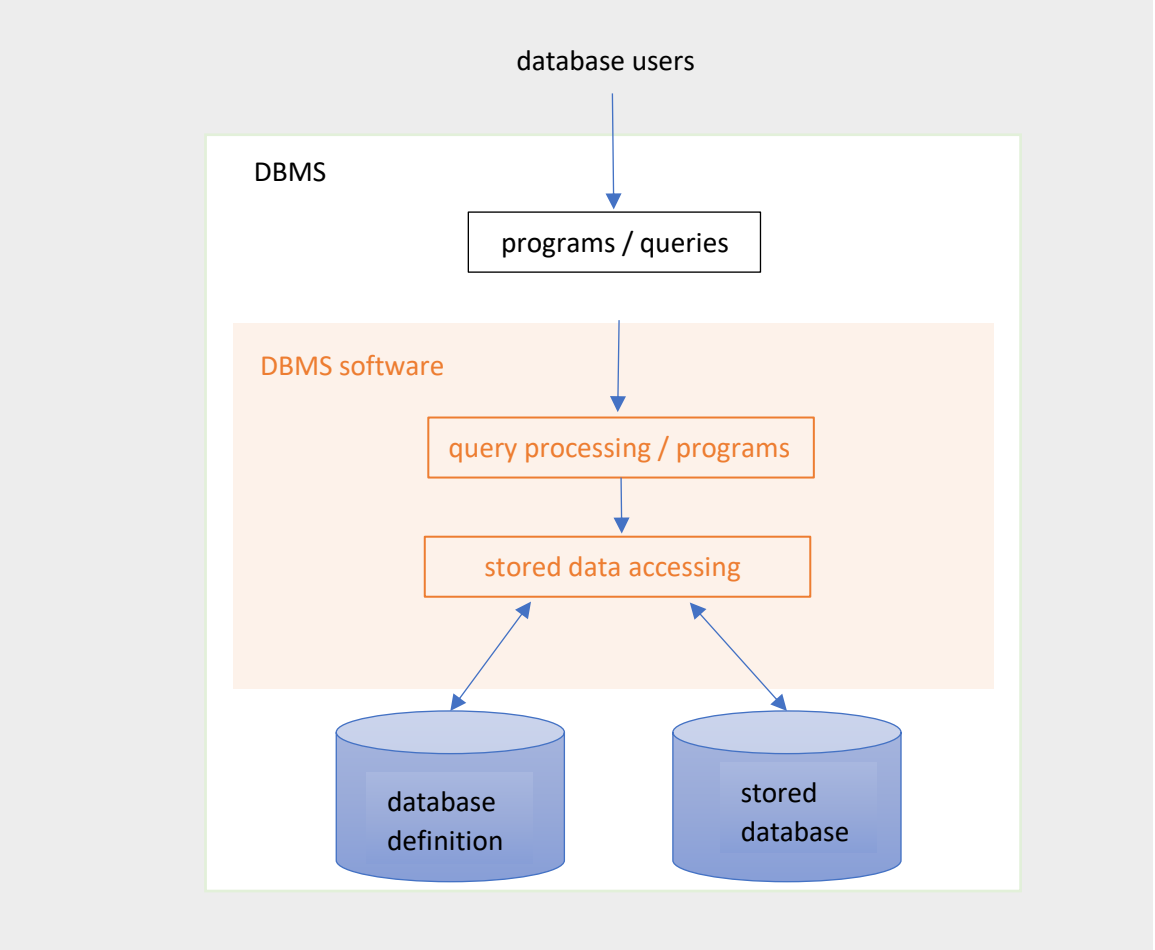

Figure 1.1: A simple view of a database system

1.2 Benefits of database approach

There are numerous benefits of using a DBMS. Some benefits are noted here.

- *Faster data sharing and improved data security*

Database management systems help in sharing data efficiently among users. It keeps data securely using different protection and security features.

 Information integration

Information integration means joining the information contained in many related databases to make it into a single one. If a company has several divisions, and each division may have built its own database over the time. Then the goal of information integration is to build a structure on the top of the existing databases, and access and modify information from the top.

 Consistency and non-repetition of data

In absence of a database management system, the same data file may get stored in multiple places or the in the same place. File updation becomes a difficult task. Even the content of the same data file may be different at different locations. This may create confusion in the decision-making processes.

 Data complying privacy regulations

Database management systems provide a better framework for the enforcement of privacy and security policies. Data can be kept in a unified manner maintaining privacy and data security at a central location.

 Increased productivity and multiple views

DBMS empowers people to spend more time on high-value activities and strategic initiatives. The same data can be viewed by multiple users placed at different locations.

 Better decision-making

Decisions built on available data have more validity. Valid decisions lead to better management of an organization.

 Providing backup and recovery

A DBMS provides facility to recover database in the events of hardware and software failures. Also, it offers facility to take backup of databases.

 Enforcing integrity constraints

Integrity constraints are a set of rules. It is enforced to maintain data with the rules. There are various forms of integrity constraints such as domain constraint, entity constraint, referential integrity constraint and key constraint. An example of entity constraint is that *account_number* is not null. Integrity constraints ensure that the data insertion and updation performed on records ensure such rules after the operations.

1.3 Data abstraction

There are different types of users of a DBMS. It may not be suitable to present data using complex structures to all the users. Data abstraction is the reduction of a particular body of data to a simplified representation of the whole. Data abstraction can be achieved through different levels of data as given below.

The lowest level of abstraction also known as the *physical level* of data. At this level, data describes complex low-level data structures in details.

Lecture 2

Contents: File-processing system; Database languages; DBMS architectures; Database system utilities; DBMS interfaces; Types of database systems; Exercises

2.1 File-processing system

In a traditional file processing system, each user defines and implements the files needed for a specific application using a programming language. In this case, a DBMS is not available. Some characteristics of a file processing system are given below.

- Each file created is a flat file and independent of another file.
- Each file contains information for one specific application such as accounting and inventory.
- Files are designed by using a programming language such as COBOL, C, and C++.

A traditional file processing system has many limitations. A few limitations are mentioned below.

- Data are duplicated in several files.
- Data are not secured. Anybody can see the data by getting into the files.
- In absence of any control, concurrent accesses of the same data can lead to problems.
- As files are generated by using different programming languages, files are of incompatible formats. Writing applications using different files may become a difficult task.
- Decisions based on available data gradually become more difficult.

2.2 Database languages

To work with a database, we need to create a database. Database schema is specified using a set of definitions expressed by a special language called a *data-definition language* (DDL). A few MySQL DDL statements are CREATE DATABASE, CREATE FUNCTION, CREATE VIEW, ALTER TABLE, and DROP VIEW statements.

The result of complication of DDL statements is a set of tables which are stored in a special file, called data dictionary. It is data about data, also known as meta-data.

Data in a database is organized using a data model. Users can access or manipulate data using a special language called *data-manipulation language* (DML). A few MySQL DML statements are INSERT, DELETE, UPDATE, SELECT and EXCEPT statements.

Procedural DMLs are used to specify what data are needed and how to get those data. But, *declarative DMLs* require a user to specify what data are needed without specifying how to get those data. Declarative DMLs are also called nonprocedural DMLs.

A query is a statement used for retrieval of information. The portion of a DML used for information retrieval is called a *query language*. SELECT statement in MySQL is a query statement.

2.3 DBMS architectures

The architecture of DBMS packages has evolved from the times. At the beginning, there were monolithic architecture, where the architecture is a singular, large computing network with one code base that couples all the business concerns together. To make a change to this sort of application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface. The computing system is supported by a single server and many dumb terminals. This is also known as centralized DBMS.

Gradually, personal computers started coming and networking technologies started becoming cheaper. DBMS systems started to exploit the available processing power of personal computers and networking technologies. All the innovations led to client/server DBMS architectures. A logical two-tier client/server architecture is given in Fig. 2.1.

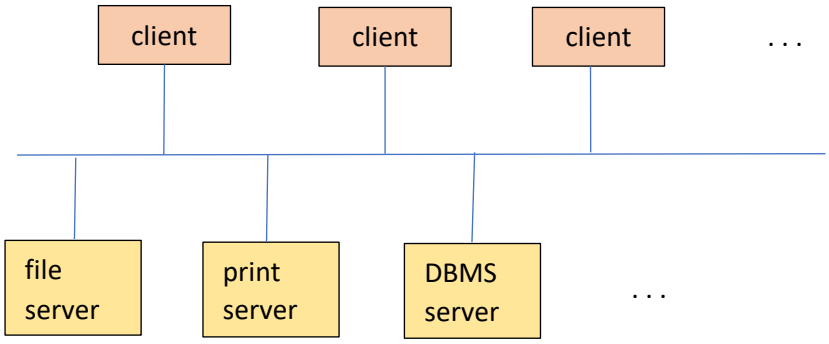

Figure 2.1: A logical two-tier client/server architecture

In a three-tier architecture, there is an intermediate layer or middle tier is called the application server or the Web server, depending on the application. The middle layer plays an intermediary role by running application programs and storing business rules that are used to access data from the database server. It can also improve database security by checking clients' credentials before forwarding requests to the database server. See Fig. 2.2.

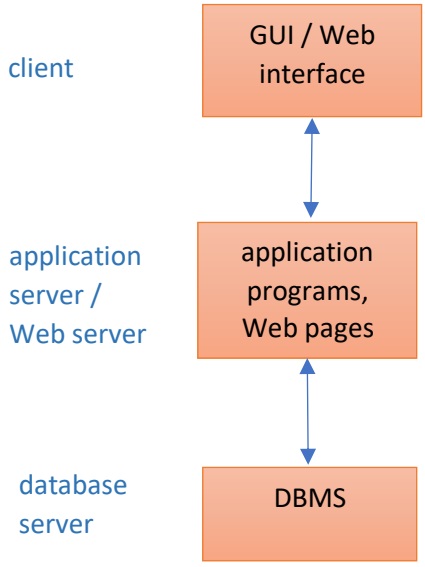

Figure 2.2: A logical three-tier client/server architecture

2.4 Database system utilities

Database utilities help the DBA manage the database system better. Utilities help performing different activities including the following.

Transferring data from one DBMS to another is done in many organizations. Some vendors offer products that perform the task by giving the existing source and target database storage schemas. Such products are also called conversion tools.

A backup utility creates a backup copy of the database in mass storage medium. The backup copy can be used to restore the database in case of disk failure.

Utility can be used to reorganize a set of database files into different file organizations and giving new access paths to improve performance.

Utilities are created to monitor database usage and provide statistics to the DBA. The statistics help in making decisions such as reorganizing files and adding / dropping indexes to improve performance.

2.5 DBMS interfaces

We discuss below a few user-friendly interfaces provided by a DBMS.

Menu-based Interfaces present the user with lists of options. Menus help to take an option without memorizing the specific commands and syntax of a query language.

A form-based interface displays a form to a user. Users can fill out the form entries to insert new data. In some cases, users fill out only certain entries and the DBMS retrieve matching data for the remaining entries.

For graphical user interfaces (GUIs), it displays a schema to the user in diagrammatic form. A user can specify a query by manipulating the diagram. In most of the cases, GUIs utilize both menus and forms. Using a pointing device like a mouse, a user selects certain parts of the schema diagram to specify a query.

Parametric users perform a small task repeatedly. For example, a bank teller performs repetitive transactions such as account deposits or withdrawals, or balance inquiries. The goal is to design and implement a parametric user interface for each known class of naive users.

DBA has a privileged account. It needs a special interface for privileged tasks such as creating accounts, setting system parameters, granting authorization, changing a schema, and reorganizing the storage structures.

2.6 Types of database systems

Let us give some characteristic of *relational database systems*. A relational database is a database based on the relational model of data. A relational model organizes data into one or more tables, also known as relations, with a unique key identifying each row. Rows are also called records or tuples. Columns are also called attributes. Many relational database systems are equipped with the option of using Structured Query Language (SQL) for querying and updating the database. A few relational database systems are named in Section 1.8 of Lecture 1.

Network model is a flexible way of representing objects and their relationships. A distinguishing feature is that a schema is viewed as a graph, where the object types are nodes and relationship types are arcs. This is not restricted to being a hierarchy or lattice. A few network database management systems are given below.

Lecture 3

Contents: Conceptual models and database design; Entities and attributes; Relationships; Entity relationship model; Entity relationship diagrams having weak entities; Some concepts; Exercises

3.1 Conceptual models and database design

Suppose we wish to develop a banking application. At the top tier of the application, we have different application programs dealing with queries, updates and reports. At the back end of the application, we have a database. Successful database design leads to a successful banking application. For designing a database, it is required to conceptualize it. Thus, conceptual modeling of database design is an important step. We discuss database structures and constraints as part of conceptual modeling. Then we need to design and test various application programs and these processes fall under software engineering.

The first step involves requirements collection and analysis. The database designers interview prospective database users to understand and document their data requirements. User requirements should be specified in detailed and complete form. Apart from data requirements, it is useful to specify functional requirements. In this case, we study various operation that are performed on the database. We use data flow diagrams, sequence diagrams, scenarios, and other techniques to specify functional requirements.

At the next step, we conceive the system using conceptual modelling. We create conceptual schema of data requirements using entity types, relationships, and constraints.

The next step is the actual implementation of the database using a commercial DBMS. Two popular implementation of data models are relational database model and object-relational database model. The conceptual schema is transformed from the high-level data model into the implementation data model. This step maps high-level data model into implementation data model.

The last step is the physical schema design phase. The internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified. In parallel with these activities, application programs are designed and tested with the help of transactions using the database as designed in an earlier stage.

3.2 Entities and attributes

An *entity* is a thing or an object in a real-world system. An entity can be described by using the attributes that is distinguishable from all other objects. For example, book and member are entities in a library system. A book can be described by attributes such as accession number, title, author1, author2, author3, edition, publisher, ISBN, and date of publication.

An attribute that can be further sub-divided logically is called a *composite attribute*. For example, *date-of-birth* attribute is collection of day, month and year; *name* attribute is collection of first name, middle name and last name; *telephone-number* attribute is a collection of country code, std code, local exchange code, and identification code.

An attribute that can assume more than one value is called *multivalued attribute*. For example, *telephone-number* is multi-valued attribute, since a person can have more than one telephone numbers; location can also be a multi-valued attribute, since a department can be located in different places.

An attribute that is neither composite nor multivalued is called an *atomic attribute*. Some examples of atomic attributes are roll number of a student, price of a product and version number of a software.

An attribute whose value can be obtained from other attributes is called a *derived attribute*. For example, *age* is a derived attribute, since it can be obtained from the date of birth.

A *superkey* is a set of one or more attributes that, taken collectively, can identify uniquely an entity in the entity set. A superkey may contain extraneous attributes. If *K* is a superkey, then so is any superset of *K*. A superkey for which no proper subset is also a superkey, is called a *candidate key*. It is possible that several distinct sets of attributes could serve as candidate keys. The *primary key* is one of the candidate keys that is chosen by the database designer as the principal means of identifying entities within the entity set.

Consider the following database table.

Student = (*rollNo*, *name*, *dateOfBirth*, *contactNo*, *major*, *class*, *address1*, *address2*), where *address1* and *address2* together forms the address of a student.

We assume that *contactNo* attribute is single valued. Some candidate keys are {*rollNo*}, {*contactNo*, *name*}, {*name*, *dateOfBirth*}, {*name*, *address1*, *address2*}.

There is a difference between primary key and unique key. A few points are given below.

∗ The main difference between the primary key and unique key is that the primary key can never have a null value while the unique key may consist of null value.

∗ In a relation, there can be only one primary key as chosen by the database designer. But, the number of unique keys in a relation may be more than one.

There are two types of entity viz., strong entity and weak entity. A *strong entity* has sufficient attributes to describe itself. For example, student is a strong entity in a university enrolment system. A student can be identified by registration number. But, a *weak entity* does not have sufficient attributes to describe itself. For example, a transaction associated with an account is a weak entity. It is described by transaction number, transaction type, transaction date and amount. For a particular account, transaction number is unique, but two transactions on different accounts may have the same transaction number. So, the above four attributes of a transaction do not describe itself uniquely.

For a weak entity, there is a set of attributes whose value differ from an instance to another instance for a given strong entity on which the weak entity is dependent. This set of attributes is called *discriminator*. The primary key of a weak entity is formed by combining the primary key of the strong entity and the discriminator of the weak entity. A discriminator is under dotted-lined.

3.3 Relationships

A relationship is an association among several entities. For example, *placed_in* relationship associates a book and a shelf, where the book is kept when it is available, in a library. In this case, *placed_in* is a binary relationship.

There are different types of binary relationship: one-to-one, one-to-many, many-to-many. Some examples of one-to-one relationship are given here. (i) relationship ”capital of” between country and capital city, (ii) relationship ”social security of” between citizen and security card, (iii) relationship ”ID card of” between employee and ID card. Examples of many-to-one relationship are (i) relationship *works_for* between employee and department, (ii) relationship *published_by* between book and publisher, (iii) relationship *designed_by* between car model and car company. Some examples of many-to-many relationships are (i) relationship *book_author* between book and author, (ii) relationship *employee_project* between employee and project, (iii) relationship *course_student* between student and course. Note that one-to-many relationship can also be viewed as a many-to-one relationship.

Consider binary relationship *works_on* between entities *Employee* and *Project* in Fig. 3.1. It is expressed using entity relationship diagram (ERD). More details about entity relationship diagrams are presented in the later sections. We note that every project has some employee(s) to work on. Thus, the participation of Project entity with this relationship is total. On the other hand, each employee may not be assigned to work on a project. Thus, participation of Employee entity with this relationship is partial. A total participation is represented by parallel lines, where as a partial participation is shown using a single line.

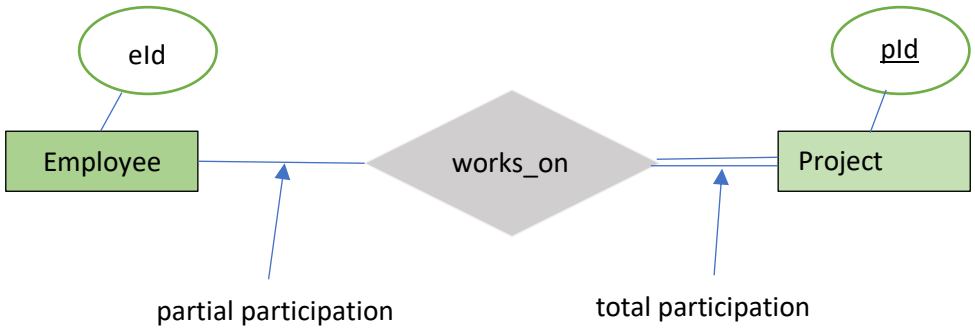

Figure 3.1: ERD showing total and partial participations

Only key fields of entities are shown, and they are underlined. Employee identification (*eid*) and project identification (*pid*) are the key attributes for Employee and Project entities respectively.

A relation is represented as a table. There are a number of columns and rows (tuples) in a relation. The number of rows changes rather more frequently that the number of columns in a relation in a database. The *degree of a relation* is the number of columns (attributes) in the relation. The *cardinality of a relation instance* is the number of tuples in the relation at a particular point of time.

Ternary relationships are often fund in many real-world systems. A few examples of ternary relationships are given as follows. (i) One can define a ternary relationship ”contract” between main actor, movie studio and movie. The cardinalities of these entities are n, 1 and n respectively. (ii) A ternary relationship ”subject taught” can exists between course, subject and teacher. The cardinalities of these entities are 1, n and n respectively. (iii) We could define a ternary relationship ”prescription” between doctor, medicine and patient. The cardinalities of these entities are 1, n and n respectively. The symbol n in this context represents more than one.