KTU NOTES

MODULE 1

1. INTRODUCTION

A **database** is a collection of related data.

Data means known facts that can be recorded and that have implicit meaning.

A database has the following implicit properties:

- A database represents some aspect of the real world, sometimes called the miniworld or the universe of discourse (DoD). Changes to the miniworld are reflected in the database.
- A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
- A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

In essence, a database has:

- Source of data.
- Interaction with real world.
- Audience interested in that data.

A **database management system (DBMS)** is a collection of programs that enables users to create and maintain a database. The DBMS is hence a general-purpose software system that facilitates the processes of

- defining,
- constructing,
- manipulating,
- sharing

databases among various users and applications.

Defining a database involves specifying the data types, structures, and constraints for the data to be stored in the database.

Constructing the database is the process of storing the data itself on some storage medium that is controlled by the DBMS.

Manipulating a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data.

Sharing a database allows multiple users and programs to access the database concurrently.

Other important functions provided by the DBMS include

- protecting the database and
- maintaining it over a long period of time.

Protection includes both system protection against hardware or software malfunction (or crashes), and security protection against unauthorized or malicious access. A typical large database may have a life cycle of many years, so the DBMS must be able to **maintain** the database system by allowing the system to evolve as requirements change over time.

The database and DBMS software are together called a **database system**.

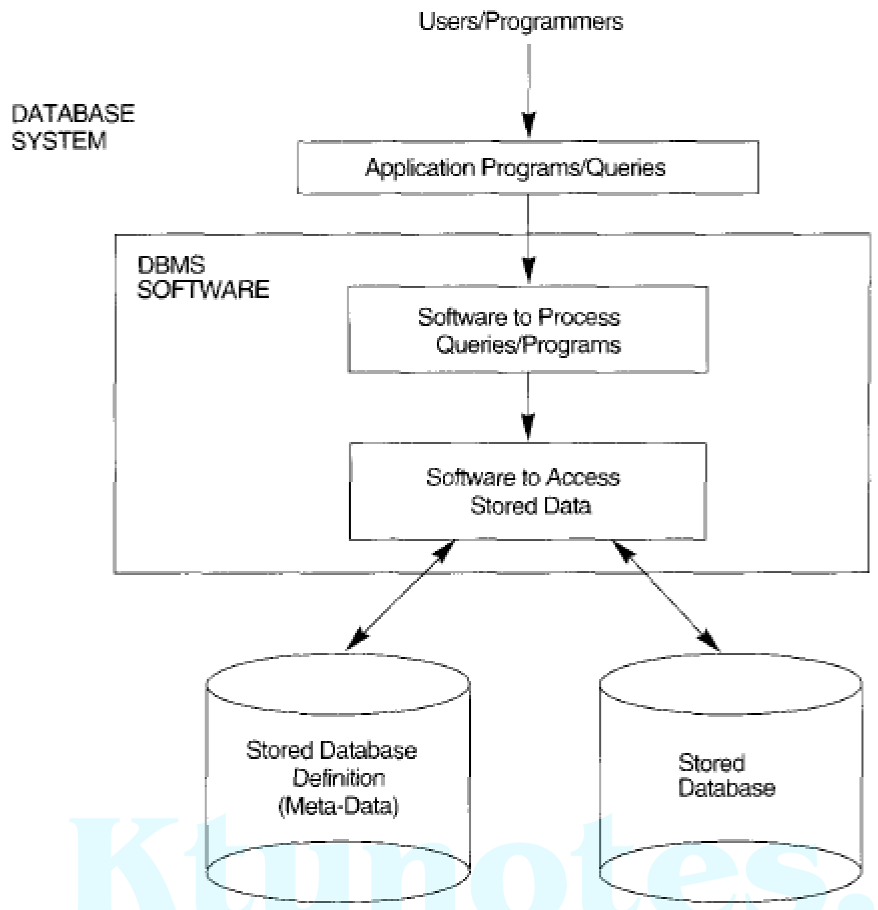
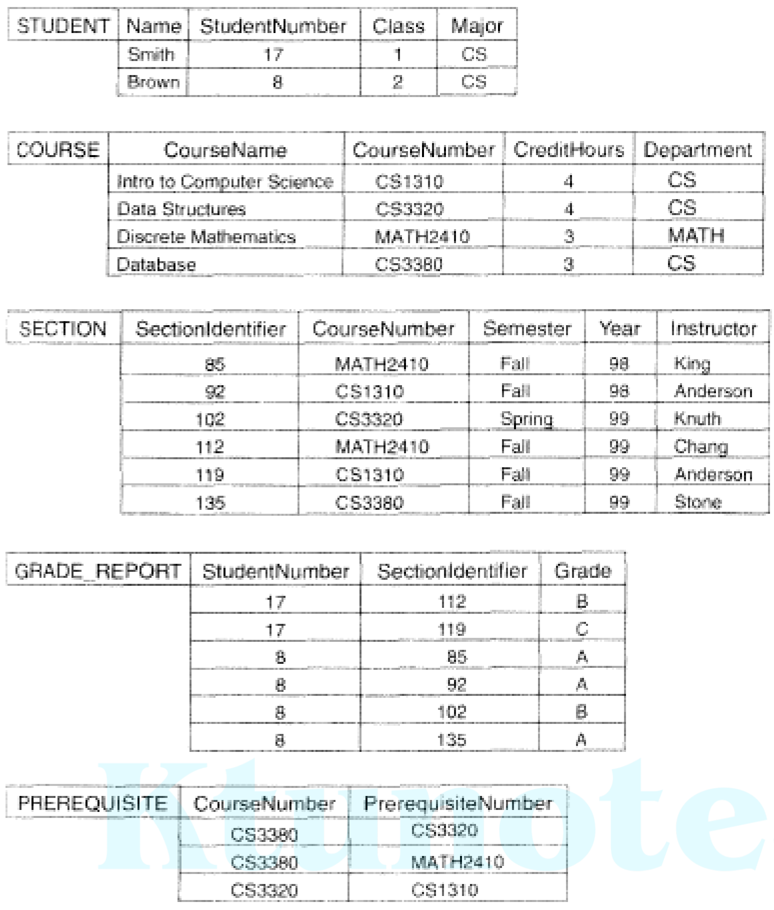

Users/Programmers
DATABASE SYSTEM
Application Programs/Queries
DBMS SOFTWARE
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

FIGURE 1.1 A simplified database system environment.

An Example



STUDENT	Name	StudentNumber	Class	Major
	Smith	17	1	CS
	Brown	8	2	CS

COURSE	CourseName	CourseNumber	CreditHours	Department
	Intro to Computer Science	CS1310	4	CS
	Data Structures	CS3320	4	CS
	Discrete Mathematics	MATH2410	3	MATH
	Database	CS3380	3	CS

SECTION	SectionIdentifier	CourseNumber	Semester	Year	Instructor
	85	MATH2410	Fall	98	King
	92	CS1310	Fall	98	Anderson
	102	CS3320	Spring	99	Knuth
	112	MATH2410	Fall	99	Chang
	119	CS1310	Fall	99	Anderson
	135	CS3380	Fall	99	Stone

GRADE_REPORT	StudentNumber	SectionIdentifier	Grade
	17	112	B
	17	119	C
	8	85	A
	8	92	A
	8	102	B
	8	135	A

PREREQUISITE	CourseNumber	PrerequisiteNumber
	CS3380	CS3320
	CS3380	MATH2410
	CS3320	CS1310

FIGURE 1.2 A database that stores student and course information.

2. CHARACTERISTICS OF DATABASE APPROACH

A number of characteristics distinguish the database approach from the traditional approach of programming with files. In traditional file processing, each user defines and implements the files needed for a specific software application as part of programming the application. There is redundancy in defining and storing data which results in wasted storage space and in redundant efforts to maintain common data up to date.
In the database approach, a single repository of data is maintained that is defined once and then is accessed by various users. The main characteristics of the database approach versus the file-processing approach are the following:

- **Self-describing nature of a database system**
  The database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the **DBMS catalog**, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called **meta-data**, and it describes the structure of the primary database.
  A general-purpose DBMS software package is not written for a specific database application, and hence it must refer to the catalog to know the structure of the files in a specific database, such as the type and format of data it will access. In traditional file

processing, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs.

- **Insulation between programs and data, and data abstraction**
  In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access this file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. This is called property **program-data independence**.

Data Item Name	Starting Position in Record	Length in Characters (bytes)
Name	1	30
StudentNumber	31	4
Class	35	4
Major	39	4

FIGURE 1.3 Internal storage format for a STUDENT record.

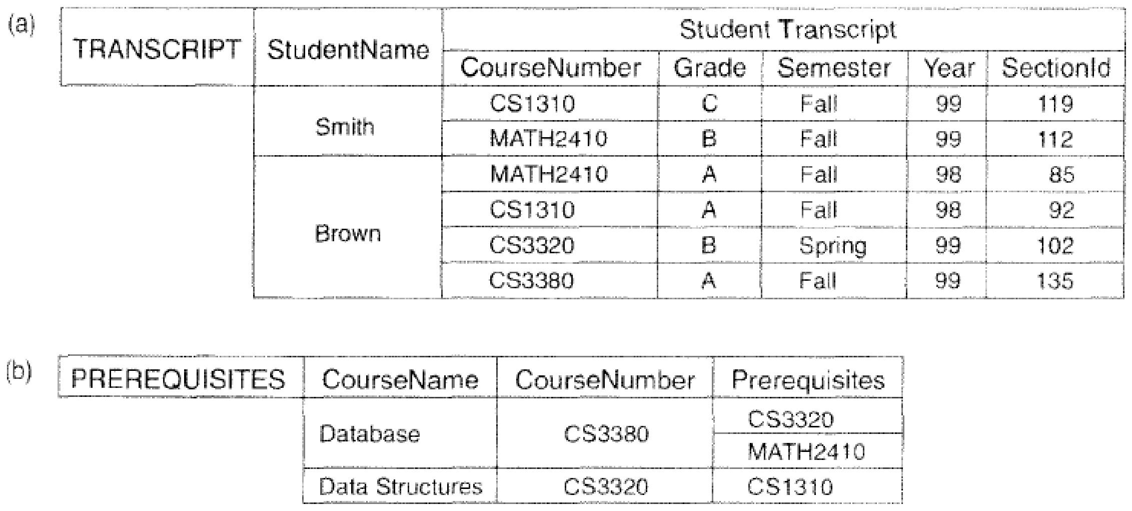
Users can define a set of operations on a data in some types of database system. An **operation** (also called a function or method) is specified in two parts.

* The **interface** (or signature) of an operation includes the operation name and the data types of its arguments (or parameters).

* The **implementation** (or method) of the operation is specified separately and can be changed without affecting the interface.

User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed **program-operation independence.**
The characteristic that allows program-data independence and program-operation independence is- called **data abstraction**. A DBMS provides users with a **conceptual representation** of data that does not include many of the details of how the data is stored or how the operations are implemented. Informally, a **data model** is a type of data abstraction that is used to provide this conceptual representation. The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model hides storage and implementation details that are not of interest to most database users.

- **Support of multiple views of the data**
  A database typically has many users, each of whom may require a different perspective or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored. Some users may not need to be aware of whether the data they refer to is stored or derived. A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.


(a)

TRANSCRIPT	StudentName	Student Transcript				
		CourseNumber	Grade	Semester	Year	SectionId
	Smith	CS1310	C	Fall	99	119
		MATH2410	B	Fall	99	112
	Brown	MATH2410	A	Fall	98	85
		CS1310	A	Fall	98	92
		CS3320	B	Spring	99	102
		CS3380	A	Fall	99	135

(b)

PREREQUISITES	CourseName	CourseNumber	Prerequisites
	Database	CS3380	CS3320
			MATH2410
	Data Structures	CS3320	CS1310

IGURE 1.4 Two views derived from the database in Figure 1.2. (a) The STUDENT TRANSCRIPT view.

- **Sharing of data and multiuser transaction processing**
  A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database. The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct. A transaction is an executing program or process that includes one or more database accesses, such as reading or updating of database records. Each transaction is supposed to execute a logically correct database access if executed in its entirety without interference from other transactions.

3. ACTORS ON THE SCENE

* **Database administrators:**
In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the database administrator (DBA). The DBA is responsible for authorizing access to the database, for coordinating and monitoring its use, and for acquiring software and hardware resources as needed. The DBA is accountable for problems such as breach of security or poor system response time.

* **Database designers:**
Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements, and to come up with a design that meets these requirements.

* **End users**
End users are the people whose jobs require access to the database for querying, updating, and generating reports, the database primarily exists for their use.
There are several categories of end users:

• **Casual end users** occasionally access the database, but they may need different information each time.

• **Naive or parametric end users** make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates-called canned transactions-that have been carefully programmed and tested.

• **Sophisticated end users** include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS so as to implement their applications to meet their complex requirements.

• **Stand-alone users** maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces.

* **System analysts and application programmers**

- **System analysts** determine the requirements of end users, especially naive and parametric end users, and develop specifications for canned transactions that meet these requirements.
- **Application programmers** implement these specifications as programs; then they test, debug, document, and maintain these canned transactions.

4. WORKERS BEHIND THE SCENE

Some others are associated with the design, development, and operation of the DBMS software and system environment. They are:

- **DBMS system designers and implementers** are persons who design and implement the DBMS modules and interfaces as a software package.
- **Tool developers** include persons who design and implement tools-the software packages that facilitate database system design and use and that help improve performance.
- **Operators and maintenance personnel** are the system administration personnel who are responsible for the actual running and maintenance of the hardware and software environment for the database system.

5. ADVANTAGES OF DBMS

The advantages of using DBMS are as follows:

1. **Controlling Redundancy:**

Redundancy in storing the same data multiple times leads to several problems such as:

- There is the need to perform a single logical update-such as entering data on a new student-multiple times: once for each file where student data is recorded. This leads to *duplication of effort.*

- *Storage space* is *wasted,* when the same data is stored repeatedly, and this problem may be serious for large databases.
- Files that represent the same data may become *inconsistent.*
  Thus in practice, it is sometimes necessary to use **controlled redundancy** for improving the performance of queries.

(a)

GRADE_REPORT	StudentNumber	StudentName	SectionIdentifier	CourseNumber	Grade
	17	Smith	112	MATH2410	B
	17	Smith	119	CS1310	C
	8	Brown	85	MATH2410	A
	8	Brown	92	CS1310	A
	8	Brown	102	CS3320	B
	8	Brown	135	CS3380	A

(b)

GRADE_REPORT	StudentNumber	StudentName	SectionIdentifier	CourseNumber	Grade
	17	Brown	112	MATH2410	B

FIGURE 1.5 Redundant storage of StudentName and CourseNumber in GRADE_REPORT. (a) Consistent data. (b) Inconsistent record.

2. Restricted Unauthorized Access

When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database. For example, financial data is often considered confidential, and hence only authorized persons are allowed to access such data. In addition, some users may be permitted only to retrieve data, whereas others are allowed both to retrieve and to update.

A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and to specify account restrictions. The DBMS should then enforce these restrictions automatically.

3. Providing Persistent Storage for Program Objects

Databases can be used to provide persistent storage for program objects and data structures.This is one of the main reasons for object-oriented database systems. The values of program variables are discarded once a program terminates, unless the programmer explicitly stores them in permanent files, which often involves converting these complex structures into a format suitable for file storage.

The persistent storage of program objects and data structures is an important function of database systems. Traditional database systems often suffered from the so called **impedance mismatch problem,** since the data structures provided by the DBMS were incompatible with the programming language's data structures.

4. Providing Storage Structures for Efficient Query Processing

Database systems must provide capabilities for efficiently executing queries and updates. Because the database is typically stored on disk, the DBMS must provide specialized data structures to speed up disk search for the desired records. Auxiliary files called **indexes** are used for this purpose. Indexes are typically based on tree data structures or hash data structures, suitably modified for disk search. In order to process the database records needed by a particular query, those records must be

copied from disk to memory. Hence, the DBMS often has a **buffering** module that maintains parts of the database in main memory buffers.
The **query processing and optimization** module of the DBMS is responsible for choosing an efficient query execution plan for each query based on the existing storage structures.

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

6. **Providing Multiple User Interfaces**
Because many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces. These include query languages for casual users, programming language interfaces for application programmers, forms and command codes for parametric users, and menu-driven interfaces and natural language interfaces for stand-alone users.

7. **Representing Complex Relationships among Data**
A DBMS must have the capability to represent a variety of complex relationships among the data as well as to retrieve and update related data easily and efficiently.

8. **Enforcing Integrity Constraints**
Most database applications have certain integrity constraints that must hold for the data. A DBMS should provide capabilities for defining and enforcing these constraints. The simplest type of integrity constraint involves specifying a data type for each data item.
These constraints are derived from the meaning or semantics of the data and of the miniworld it represents. It is the database designers' responsibility to identify integrity constraints during database design. Some constraints can be specified to the DBMS and automatically enforced. Other constraints may have to be checked by update programs or at the time of data entry.

9. **Permitting Inferencing and Actions Using rules**
Some database systems provide capabilities for defining *deduction rules* for inferencing new information from the stored database facts. Such systems are called deductive database systems. For example, there may be complex rules in the miniworld application for determining when a student is on probation. These can be specified *declaratively* as rules, which when compiled and maintained by the DBMS can determine all students on probation. In a traditional DBMS, an explicit *procedural prof-,Jmm code* would have to be written to support such applications. But if the

miniworld rules change, it is generally more convenient to change the declared deduction rules than to recode procedural programs. More powerful functionality is provided by active database systems, which provide active rules that can automatically initiate actions when certain events and conditions occur.

10. Additional Implications of Using the Database Approach

- **Potential for Enforcing Standards:** The database approach permits the DBA to define and enforce standards among database users in a large organization.
- **Reduced Application Development Time:** A prime selling feature of the database approach is that eveloping a new application-such as the retrieval of certain data from the database for printing a new report-takes very little time.
- **Flexibility:** It may be necessary to change the structure of a database as requirements change.
- **Availability of Up-to-Date Information:** A DBMS makes the database available to all users.
- **Economies of Scale**: The DBMS approach permits consolidation of data and applications, thus reducing the amount of wasteful overlap between activities of data processing personnel in different projects or departments.

6. WHEN NOT TO USE DBMS

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

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

Additional problems may arise if the database designers and DBA do not properly design the database or if the database systems applications are not implemented properly. Hence, it may be more desirable to use regular files under the following circumstances:
• The database and applications are simple, well defined, and not expected to change.
• There are stringent real-time requirements for some programs that may not be met because of DBMS overhead.
• Multiple-user access to data is not required.

7. DATA MODELS, SCHEMAS AND INSTANCES

- **Data Abstraction** generally refers to the suppression of details of data organisation and storage and the highlighting of the essential features for an improved understanding of data.
- **Data model** is a collection of concepts that can be used to describe the structure of a database.

7.1 Categories of Data Models

- **High-level** or **conceptual data models** provide concepts that are close to the way many users perceive data, whereas **low-level** or **physical data models** provide concepts that describe the details of how data is stored in the computer.

- **Representational (or implementation) data models**, which provide concepts that may be understood by end users but that are not too far removed from the way data is organized within the computer. Representational data models hide some details of data storage but can be implemented on a computer system in a direct way.

  Representational or implementation data models are the models used most frequently commercial DBMSs. These include the widely used relational data model, as well as the so-called legacy data models-the network and hierarchical models-that have been widely used in the past. Representational data models represent data by using record structures and hence are sometimes called record-based data models.

- **Conceptual data models** use concepts such as entities, attributes, and relationships. An **entity** represents a real-world object or concept, such as an employee or a project, that is described in the database. An **attribute** represents some property of interest that further describes an entity, such as the employee's name or salary. A **relationship** among two or more entities represents an association among two or more entities.

- **Physical data models** describe how data is stored as files in the computer by representing information such as record formats, record orderings, and access paths. An **access path** is a structure that makes the search for particular database records efficient. An **index** is an example of an access path that allows direct access to data using an index term or a keyword.

7.2 Schemas, Instances and Database State

The description of a database is called the **database schema**, which is specified during database design and is not expected to change frcquentlv. Most data models have certain conventions for displaying schemas as diagrams." A displayed schema is called a **schema diagram.** A schema diagram displays only *some aspects* of a schema, such as the names of record types and data items, and some types of constraints. We call each object in the schema-such as STUDENT or COURSE-a **schema construct**.

The actual data in a database may change quite frequently. The data in the database at a particular moment in time is called a **database state** or **snapshot**. It is also called the *current* set of **occurrences** or **instances** in the database. In a given database state, each schema construct has its own *current set* of instances; for example, the STUDENT construct will contain the set of individual student entities (records) as its instances.

The distinction between database schema and database state is very important. When we **define** a new database, we specify its database schema only to the DBMS. At this point, the corresponding database state is the *empty state* with no data. We get the *initial state* of the database when the database is first **populated** or **loaded** with the initial data. From then on, every time an update operation is applied to the database, we get another database state. At any point in time, the database has a *current state.* The DBMS is partly responsible for ensuring that *every* state of the database is a **valid state**-that is, a state
that satisfies the structure and constraints specified in the schema.
The DBMS stores the descriptions of the schema constructs and constraints-also called the **meta-data**-in the DBMS catalog so that DBMS software can refer to the schema whenever it needs to. The schema is sometimes called the **intension**,
and a database state an **extension** of the schema.

8. THREE SCHEMA ARCHITECTURE AND DATA INDEPENDENCE

Three of the four important characteristics of the database approach, are
(1) insulation of program and data (program-data and program-operation independence),
(2) support of multiple user views,
(3) use of a catalog to store the database description (schema).

8.1 Three Schema Architecture :

It is an architecture for database systems that was proposed to help achieve and visualize these characteristics.
The goal of the three-schema architecture is to separate the user applications and the physical database.
In this architecture, schemas can be defined at the following three levels:
1. The **internal level has an internal schema**, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. The **conceptual level has a conceptual schema**, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data
types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This *implementation conceptual schema* is often based on a *conceptual schema design* in a high-level data model.
3. **The external or view level** includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group.Each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high level data model.

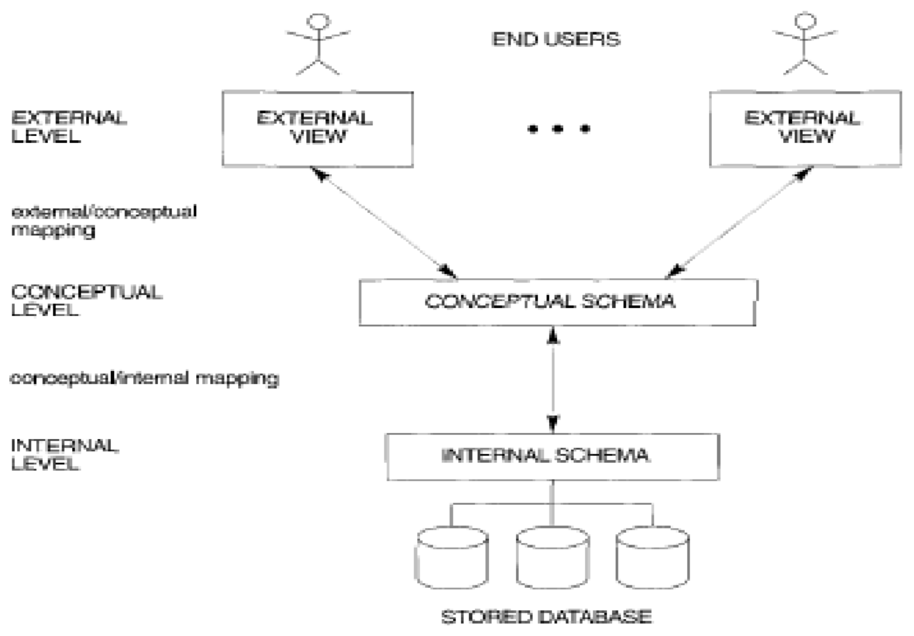


FIGURE 2.2 The three-schema architecture.

The three-schema architecture is a convenient tool with which the user can visualize the schema levels in a database system. Most DBMSs do not separate the three levels completely, but support the three-schema architecture to *some* extent. Some DBMSs may include physical-level details in the conceptual schema. In most DBMSs that support user views, external schernas are specified in the same data model that describes the conceptual-level information. Some DBMSs allow different data models to be used at the conceptual and external levels.

In a DBMS based on the three-schema architecture, each user group refers only to its own external schema. Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view.

The processes of transforming requests and results between levels are called **mappings**. These mappings may be time-consuming, so some DBMSs-especially those that are meant to support small databases-do not support external views.

8.2 Data Independence :

The three-schema architecture can be used to explain the concept of data independence which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level.

1**. Logical data independence** is the capacity to change the conceptual schema without having to change external schernas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item). In the last case, external schemas that refer only to the remaining data should not be affected. Only the view definition and the mappings need be changed in a DBMS that supports logical data independence.

2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well.

Changes to the internal schema may be needed because some physical files had to be reorganized-for example, by creating additional access structures-to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.

9. DATABASE LANGUAGES AND INTERFACES

9.1 DBMS Languages

In many DBMSs where no strict separation of levels is maintained, one language, called the **data definition language (DDL)**, is used by the DBA and by database designers to define both schemas. The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog.
Storage definition language (SDL), is used to specify the internal schema. The mappings between the two schemas may be specified in either one of these languages. Storage definition is typically kept separate, since it is used for defining physical storage structures to fine-tune the performance of the database system, which is usually done by the DBA staff.
For a true three-schema architecture, we would need a third language, the **view definition language (VDL)**, to specify user views and their mappings to the conceptual schema, but in most DBMSs the DDL is used to define both conceptual and external schemas.
Typical manipulations include retrieval, insertion, deletion, and modification of the data. The DBMS provides a set of operations or a language called the **data manipulation language (DML)** for these purposes.
There are two main types of **DMLs**.
A **high-level or nonprocedural DML** can be used on its own to specify complex database operations in a concise manner. Many DBMSs allow high-level DML statements either to be entered interactively from a display monitor or terminal or to be embedded in a general-purpose programming language. DML statements must be identified within the program so that they can be extracted by a precompiler and processed by the DBMS. High-level DMLs, such as SQL, can specify and retrieve many records in a single DML statement and are hence called **set-at-a-time or set-oriented DMLs**. A query in a high-level DML often specifies *which* data to retrieve rather than *how* to retrieve it; hence, such languages are also called **declarative**. On the other hand, a high-level DML used in a stand-alone interactive manner is called a **query language**.

A **low-level or procedural DML** *must* be embedded in a general-purpose programming language. This type of DML typically retrieves individual records or objects from the database and processes each separately. Hence, it needs to use programming language constructs, such as looping, to retrieve and process each record from a set of records. Low-level DMLs are also called **record-at-a-time** DMLs because of this property.
Whenever DML commands, whether high *level* or low level, are embedded in a general-purpose programming language, that language is called the host language and the DML is called the **data sublanguage**."
In general, both retrieval and update commands of a high-level DML may be used interactively and are hence considered part of the query language.
Casual end users typically use a high-level query language to specify their requests, whereas programmers use the DML in its embedded form. For naive and parametric users, there

usually are user-friendly interfaces for interacting with the database; these can also be used by casual users or others who do not want to learn the details of a high-level query language.

9.2 DBMS Interfaces

User-friendly interfaces provided by a DBMS may include the following:

Menu-Based Interfaces for Web Clients or Browsing:
These interfaces present the user with lists of options, called menus that lead the user through the formulation of a request. Menus do away with the need to memorize the specific commands and syntax of a query language; rather, the query is composed step by step by picking options from a menu that is displayed by the system.
Pull-down menus are a very popular technique in **Web-based user interfaces**. They are also often used in **browsing interfaces**, which allow a user to look through the contents of a database in an exploratory and unstructured manner.

Forms-Based Interfaces:
A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert new data, or they fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries. Forms are usually designed and programmed for naive users as interfaces to canned transactions. Many DBMSs have **forms specification languages**, which are special languages that help programmers, specify such forms. Some systems have utilities that define a form by letting the end user interactively construct a sample form on the screen.

Graphical User Interfaces:
A graphical interface (GUI) typically displays a schema to the user in diagrammatic form. The user can then specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms. Most GUIs use a pointing device, such as a mouse, to pick certain parts of the displayed schema diagram.

Natural Language Interfaces:
These interfaces accept requests written in English or some other language and attempt to "understand" them. A natural language interface usually has its own "schema," which is similar to the database conceptual schema, as well as a dictionary of important words. The natural language interface refers to the words in its schema, as well as to the set of standard words in its dictionary, to interpret the request. If the interpretation is successful, the interface generates a high-level query corresponding to the natural language request and submits it to the DBMS for processing; otherwise, a dialogue is started with the user to clarify the request.

Interfaces for Parametric Users:
Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. Systems analysts and programmers design and implement a special interface for each known class of naive users. Usually, a small set of abbreviated commands is included, with the goal of minimizing the number of keystrokes required for each request.

Interfaces for the DBA:
Most database systems contain privileged commands that can be used only by the DBA's staff. These include commands for creating accounts, setting system parameters, granting

account authorization, changing a schema, and reorganizing the storage structures of a database.

10. DATABASE SYSTEM ENVIRONMENT

10.1 DBMS COMPONENT MODULES

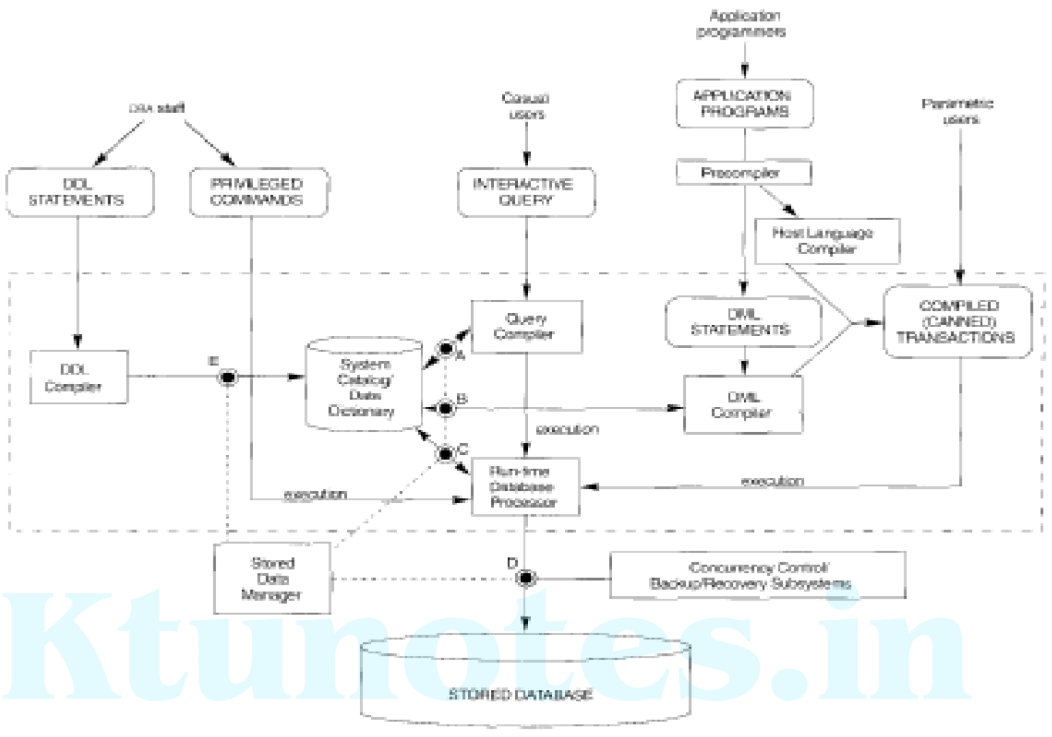


FIGURE 2.3 Component modules of a DBMS and their interactions.

The database and the DBMS catalog are usually stored on disk. Access to the disk is controlled primarily by the **operating system** (OS), which schedules disk input/output. A higher-level **stored data manager** module of the DBMS controls access to DBMS information that is stored on disk, whether it is part of the database or the catalog. The dotted lines and circles marked A, B, C, D, and E in Figure illustrates accesses that are under the control of this stored data manager. The stored data manager may use basic os services for carrying out lowlevel data transfer between the disk and computer main storage, but it controls other aspects of data transfer, such as handling buffers in main memory. Once the data is in main memory buffers, it can be processed by other DBMS modules, as well as by application programs. Some DBMSs have their own **buffer manager module**, while others use the os for handling the buffering of disk pages.

The DDL compiler processes schema definitions, specified in the DOL, and stores descriptions of the schemas (meta-data) in the DBMS catalog. The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints, in addition to many other types of information that are needed by the DBMS modules. DBMS software modules then look up the catalog information as needed.

The **runtime database processor** handles database accesses at runtime; it receives retrieval or update operations and carries them out on the database. Access to disk goes through the stored data manager, and the buffer manager keeps track of the database pages in memory. The **query compiler** handles high-level queries that are entered interactively. It parses, analyzes, and compiles or interprets a query by creating database access code, and then generates calls to the runtime processor for executing the code.

The **precompiler** extracts DML commands from an application program written in a host programming language. These commands are sent to the DML compiler for compilation into object code for database access. The rest of the program is sent to the host language compiler. The object codes for the DML commands and the rest of the program are linked, forming a canned transaction whose executable code includes calls to the runtime database processor.

It is now common to have the **client program** that accesses the DBMS running on a separate computer from the computer on which the database resides. The former is called the **client computer**, and the latter is called the **database server**. In some cases, the client accesses a middle computer, called the **application server**, which in turn accesses the database server.

The DBMS interacts with the operating system when disk accesses-to the database or to the catalog-are needed. If the computer system is shared by many users, the os will schedule DBMS disk access requests and DBMS processing along with other processes. On the other hand, if the computer system is mainly dedicated to running the database server, the DBMS will control main memory buffering of disk pages. The DBMS also interfaces with compilers for general-purpose host programming languages, and with application servers and client programs running on separate machines through the system network interface.

10.2 DATABASE SYSTEM UTILITIES

DBMSs have database utilities that help the DBA in managing the database system. Common utilities have the following types of functions:

- **Loading**: A loading utility is used to load existing data files-such as text files or sequential files-into the database. Usually, the current (source) format of the data file and the desired (target) database file structure are specified to the utility, which then automatically reformats the data and stores it in the database. Some vendors are offering products that generate the appropriate loading programs, given the existing source and target database storage descriptions (internal schemas). Such tools are also called **conversion tools**.
- **Backup:** A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape. The backup copy can be used to restore the database in case of catastrophic failure. Incremental backups are also often used, where only changes since the previous backup are recorded. Incremental backup is more complex but saves space.
- **File reorganization:** This utility can be used to reorganize a database file into a different file organization to improve performance.
- **Performance monitoring:** Such a utility monitors database usage and provides statistics to the DBA. The DBA uses the statistics in making decisions such as whether or not to reorganize files to improve performance.

Other utilities may be available for sorting files, handling data compression, monitoring access by users, interfacing with the network, and performing other functions.

10.3 TOOLS, APPLICATION ENVIRONMENTS AND COMMUNICATION FACILITIES

Other tools are often available to database designers, users, and DBAs. CASE tools are used in the design phase of database systems. Another tool that can be quite useful in large organizations is an expanded **data dictionary (or data repository)** system. In addition to storing catalog information about schemas and constraints, the data dictionary stores other information, such as design decisions, usage standards, application program descriptions, and user information. Such a system is also called an **information repository**.

This information can be accessed directly by users or the DBA when needed. A data dictionary utility is similar to the DBMS catalog, but it includes a wider variety of information and is accessed mainly by users rather than by the DBMS software.

Application development environments, such as the PowerBuilder (Sybase) or JBuilder (Borland) system, are becoming quite popular. These systems provide an environment for developing database applications and include facilities that help in
many facets of database systems, including database design, CUI development, querying and updating, and application program development.

The DBMS also needs to interface with communications software, whose function is to allow users at locations remote from the database system site to access the database through computer terminals, workstations, or their local personal computers. These are connected to the database site through data communications hardware such as phone lines, long-haul networks, local area networks, or satellite communication devices. The integrated DBMS and data communications system is called a **DB/DC system**. In addition, some distributed DBMSs are physically distributed over multiple machines. In this case, communications networks are needed to connect the machines. These are often local area networks (LANs), but they can also be other types of networks.

11. CLASSIFICATION OF DATABASE MANAGEMENT SYSTEM

Several criteria are normally used to classify DBMSs.

1. **The data model on which the DBMS is based:** The main data model used in many current commercial DBMSs is the **relational data model**. The **object data model** was implemented in some commercial systems but has not had widespread use. Many legacy (older) applications still run on database systems based on the **hierarchical** and **network** data models. The relational DBMSs are evolving continuously, and, in particular, have been incorporating many of the concepts that were developed in object databases. This has led to a new class of DBMSs called **object-relational** DBMSs. We can hence categorize DBMSs based on the data model: relational, object, object-relational, hierarchical, network, and other.

2. **The number of users supported by the system: Single-user systems** support only one user at a time and are mostly used with personal computers**. Multiuser systems**, which include the majority of DBMSs, support multiple users concurrently.
3. **The number of sites over which the database is distributed:** A DBMS is centralized if the data is stored at a single computer site. A **centralized DBMS** can support multiple users, but the DBMS and the database themselves reside totally at a single computer site. A **distributed DBMS** (DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network. **Homogeneous DDBMS** use the same DBMS software at multiple sites. A recent trend is to develop software to access several autonomous pre-existing databases stored under **heterogeneous DBMS**. This leads to a **federated DBMS** (or multi database system), in which the participating DBMSs are loosely coupled and have a degree of local autonomy. Many DBMSs use a client-server architecture.
4. **The cost of the DBMS:** The majority of DBMS packages cost between $10,000 and $100,000. Single-user low-end systems that work with microcomputers cost between $100 and $3000. At the other end of the scale, a few elaborate packages cost more than $100,000.
5. We can classify a DBMS on the basis of the types of access path options for storing files. One well-known family of DBMSs is based on inverted file structures.
6. **DBMS can be general purpose or special purpose:** When performance is a primary consideration, a special-purpose DBMS can be designed and built for a specific application such a system cannot be used for other applications without major hanges. Many airline reservations and telephone directory systems developed in the past are special purpose DBMSs. These fall into the category of **online transaction processing (OLTP)** systems, which must support a large number of concurrent transactions without imposing excessive delays.

The **object data model** defines a database in terms of objects, their properties, and their operations. Objects with the same structure and behavior belong to a class, and classes are organized into hierarchies (or acyclic graphs). The operations of each class are specified in terms of predefined procedures called methods. Relational DBMSs have been extending their models to incorporate object database concepts and other capabilities, these systems are referred to as **object-relational** or **extended relational systems**. Two older, historically important data models, now known as **legacy data models**, are the **network** and **hierarchical** models. The network model represents data as record types and also represents a limited type of l:N relationship, called a **set type**.

The **network model**, also known as the CODASYL DBTG model, l3 has an associated record-at-a-time language that must be embedded in a host programming language. The **hierarchical model** represents data as hierarchical tree structures. Each hierarchy represents a number of related records. There is no standard language for the hierarchical model, although most hierarchical DBMSs have record-at-a-time languages.

The **XML** (eXtended Markup Language) model, now considered the standard for data interchange over the Internet, also uses hierarchical tree structures. It combines database concepts with concepts from document representation models. Data is represented as elements, which can be nested to create complex hierarchical structures. This model conceptually resembles the object model, but uses different terminology.

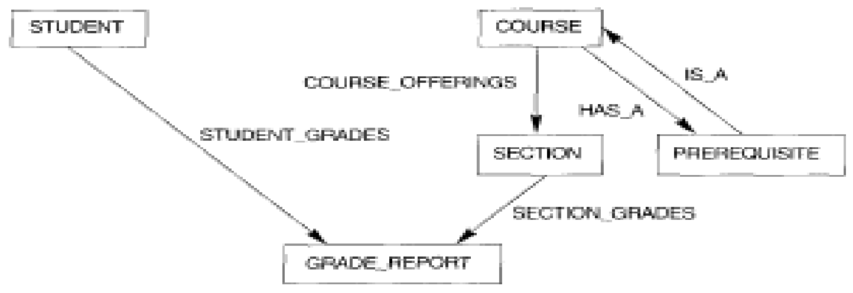


FIGURE 2.8 The schema of Figure 2.1 in network model notation

12. DFSDFSDF