CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test 1 – August 2023

Sub:	Database Management System							Sub Code:	22MCA21
Date:	01-08-2023	Duration:	90 min's	Max Marks:	50	Sem:	II	Branch:	MCA

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

	PART I	MARKS	OBE	
			CO	RBT
1	List and explain the advantages of using the DBMS approach. **OR**	[10]	CO1	L1
2	Explain about actors on the scene and workers behind the scene.	[10]	CO1	L1
3	**PART II** Explain with necessary diagram the Database Component Modules. **OR**	[10]	CO1	L1
4	Define data model and snapshot. Explain the different categories of data model.	[10]	CO1	L1

5	**PART III** What is an attribute? Explain the different types of attributes that occur in ER-Model with Examples. Write their corresponding notations? **OR**	[10]	CO1	L1
6	Explain SQL data definition and data types in brief and explain different CREATE and ALTER command.	[10]	CO2	L2
7	PART IV Create the table named 'employee' with the attributes: empno, empname, dept, designation, salary, doj, place. Write SQL statements for the following: i. Display all the fields of employee table. ii. Display details of employee number and their salary.iii. Display average salary of all employees. iv. Display distinct name of employees in descending order. v. Count number of employees. vi. Display the employees whose name contains second and third letters as 'um'. vii. Display salary of employee which is greater than 120000. viii. Display details of employee whose name is 'Amit' and salary greater than 50000 **OR**	[10]	CO2	L3

8	Define data independence and mapping. Explain the three-schema architecture with diagram.	[10]	CO1	L1
9	PART V List and explain the data types that are allowed for SQL attributes with example. How char data type differs from varchar? **OR**	[10]	CO2	L2
10	With a neat diagram explain the phases of database design.	[10]	CO1	L1

Solution

1. List and explain the advantages of using the DBMS approach.

1. Controlling Redundancy

2. Restricting Unauthorized Access

3. Providing Persistent Storage for Program Objects and Data Structures

4. Providing storage structures and search Techniques for efficient query processing

5. Permitting Inferencing and Actions Using Rules

6. Providing Multiple User Interfaces

7. Representing Complex Relationships Among Data

8. Enforcing Integrity Constraints

9. Providing Backup and Recovery

10. Additional Implications of the Database Approach

i. Controlling Redundancy • redundancy in storing the same data multiple times leads to several problems o there is the need to perform a single logical update o storage space is wasted when the same data is stored repeatedly o Files that represent the same data may become inconsistent

ii. Restricting Unauthorized Access • A DBMS should provide a security and authorization subsystem • most users will not be authorized to access all information in the database

iii. Providing Persistent Storage for Program Objects • Databases can be used to provide persistent storage for program objects and data structures

iv. Providing Storage Structures and Search Techniques for Efficient Query Processing • the DBMS must provide specialized data structures and search techniques to speed up disk search for the desired records – index • The DBMS often has a buffering or caching 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.

v. Providing Backup and Recovery
responsible for recovery • the database is restored to the state it was in before the transaction started executing

vi. Providing Multiple User Interfaces • a DBMS should provide a variety of user interfaces o query languages for casual users, o programming language interfaces for application programmers, o forms and command codes for parametric users, and o menu-driven interfaces and natural language interfaces for standalone users

vii. Representing Complex Relationships among Data • A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficientl

viii. Enforcing Integrity Constraints • The simplest type of integrity constraint involves specifying a data type for each data item • referential integrity constraint: specifying that a record in one file must be related to records in other files • Primary key constraint: uniqueness on data item values.

ix. Permitting inferencing and Actions Using Rules: A trigger is a form of a rule activated by updates to the table, which results in performing some additional operations to some other tables, sending messages, and so on.

x. Additional Implications of Using the Database Approach
Potential for Enforcing Standards: The DBA can enforce standards in a centralized database environment o Reduced Application Development Time: Development time using a DBMS is estimated to be one-sixth to one-fourth of that for a traditional file system o Flexibility: Modern DBMSs allow certain types of evolutionary changes to the structure of the database without affecting the stored data and the existing application programs o Availability of Up-to-Date Information: As soon as one user's update is applied to the database, all other users can immediately see this update o Economies of Scale: reduce the wasteful overlap activities thereby reducing the overall costs of operation and management.

2. Explain about actors on the scene and workers behind the scene.

Actors on the Scene

1. Database Administrators

2. Database Designers

3. End Users

4. System Analysts and Application Programmers (Software Engineers)

Database Administrators

In any organization where many persons use the same resources, there is a need for a chief administrator to oversee and manage these resources. 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.

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 generatingreports; 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. They use a sophisticated database query language to specify their requests and are typically middle- or high-levelmanagers or other occasional browsers.
- **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.

  Bank tellers check account balances and post withdrawals and deposits.

- **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- or graphics-based interfaces. An example is the user of a tax package that stores a variety of personal financial data for tax purposes.

System Analysts and Application Programmers (Software Engineers)

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. Such analysts and programmers (nowadays called **software engineers**) should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.

Workers behind the Scene

In addition to those who design, use, and administer a database, others are associated with the design, development, and operation of the DBMS *software and system environment.* These persons are typically not interested in the database itself. We call them the "workers behind the scene," and they include the following categories.

- **DBMS system designers and implementers** are persons who design and implement the DBMS modules andinterfaces as a software package. A DBMS is a complex software system that consists of many components or **modules,** including modules for implementing the catalog, query language, interface processors, data access, concurrency control, recovery, and security.
- **Tool developers** include persons who design and implement **tools**—the software packages that facilitate database system design and use, and help improve performance. Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data generation .
- **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.

3. Explain with necessary diagram the Database Component Modules.

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 stored data manager may use basic OS services for carrying out low- level data transfer between the disk and computer main storage, but it controls other aspects of data transfer, such as handling buffers in main memory.

The **DDL compiler** processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog. The catalog includes information such as the names of files, data items, storage details of each file and so on.

The **run-time database processor** handles database accesses at run time; it receives retrieval or update **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 run-time processor forexecuting the code.

The **pre-compiler** 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.

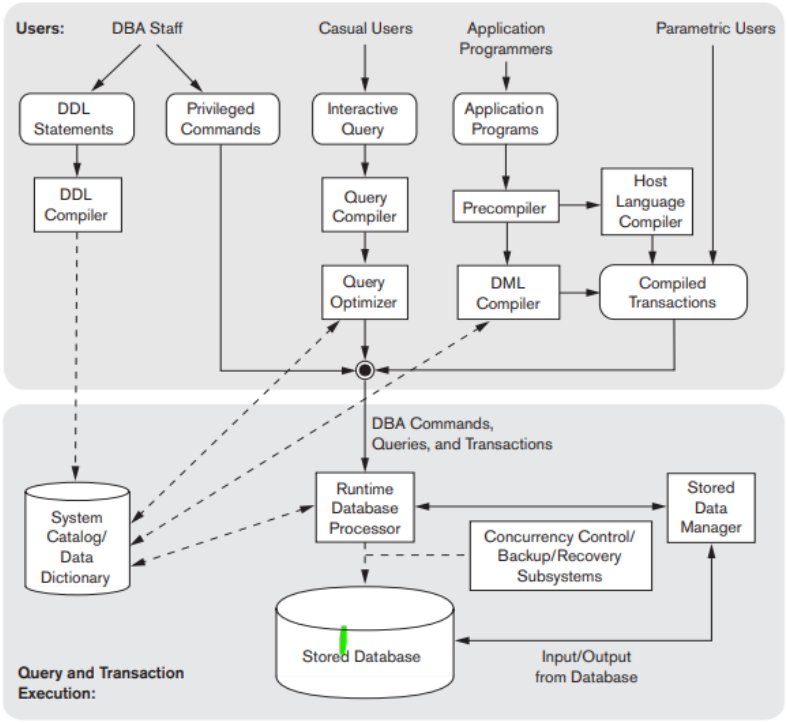


Figure 2.3
Component modules of a DBMS and their interactions.

4. Define data model and snapshot. Explain the different categories of data model.

One fundamental characteristic of the database approach is that it provides some level of data abstraction. **Data abstraction** generally refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data. One of the main characteristics of the database approach is to support data abstraction so that different users can perceive data at their preferred level of detail. A **data model**—a collection of concepts that can be used to describe the structure of a database—provides the necessary means to achieve this abstraction.[2] By *structure of a database* we mean the data types, relationships, and constraints that apply to the data. Most data models also include a set of **basic operations** for specifying retrievals and updates on the database.

In addition to the basic operations provided by the data model, it is becoming more common to include concepts in the data model to specify the **dynamic aspect** or **behavior** of a database application. This allows the database designer to specify a set of valid user-defined operations that are allowed on the database objects.[3] An example of a user-defined operation could be COMPUTE_GPA, which can be applied to a STUDENT object. On the other hand, generic operations to insert, delete, modify, or retrieve any kind of object are often included in the *basic data model operations*. Concepts to specify behavior are fundamental to object-oriented data models (see Chapter 11) but are also being incorporated in more traditional data models. For example, object-relational models (see Chapter 11) extend the basic relational model to include such concepts, among others. In the basic relational data model, there is a provision to attach behavior to the relations in the form of persistent stored modules, popularly known as stored procedures (see Chapter 13).

2.1.1 Categories of Data Models

Many data models have been proposed, which we can categorize according to the types of concepts they use to describe the database structure. **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 on the computer storage media, typically magnetic disks. Concepts provided by low-level data models are generally meant for computer specialists, not for end users. Between these two extremes is a class of **representational** (or **implementation**) **data models**,[4] which provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage. Representational data models hide many details of data storage on disk but can be implemented on a computer system directly.

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 from the miniworld 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 the entities, for example, a works-on relationship between an employee and a project. Chapter 7 presents the **Entity-Relationship model**—a popular high-level conceptual data model. Chapter 8 describes additional abstractions used for advanced modeling, such as generalization, specialization, and categories (union types).

Representational or implementation data models are the models used most frequently in traditional 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. Part 2 is devoted to the relational data model, and its constraints, operations and languages.[5] The SQL standard for relational databases is described in Chapters 4 and 5. Representational data models represent data by using record structures and hence are sometimes called **record-based data models**.

We can regard the **object data model** as an example of a new family of higher-level implementation data models that are closer to conceptual data models. A standard for object databases called the ODMG object model has been proposed by the Object Data Management Group (ODMG). We describe the general characteristics of object databases and the object model proposed standard in Chapter 11. Object data models are also frequently utilized as high-level conceptual models, particularly in the software engineering domain.

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. We discuss physical storage techniques and access structures in Chapters 17 and 18. An **index** is an example of an access path that allows direct access to data using an index term or a keyword. It is similar to the index at the end of this book, except that it may be organized in a linear, hierarchical (tree-structured), or some other fashion.

5. What is an attribute? Explain the different types of attributes that occur in ER-Model with Examples. Write their corresponding notations?

Attributes : Each entity has **attributes**—the particular properties that describe it. For example, an employee entity may be described by the employee's name, age, address, salary, and job.

A particular entity will have a **value** for each of its attributes. The attribute values that describe each entity become a major part of the data stored in the database.

Ex: 1)The employee entity e1 has four attributes: Name, Address, Age, and HomePhone; their values are "John Smith," "2311 Kirby, Houston, Texas 77001," "55," and "713-749-2630," respectively.
Several types of attributes occur in the ER model:

1) *simple* versus *composite*
2) *single-valued* versus *multi valued*
3) *stored*versus *derived.*

1) **Composite attributes** can be divided into smaller subparts, which represent more basic attributes with independent meanings.

Ex: Address attribute of the employee entity can be sub-divided into StreetAddress, City, State, and Zip with thevalues "2311 Kirby," "Houston," "Texas," and "77001."

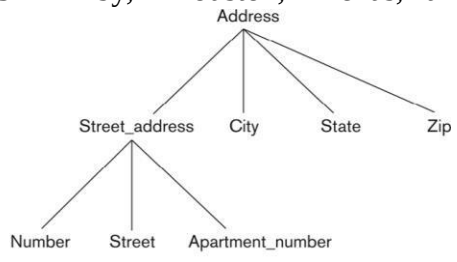


Figure 3.4
A hierarchy of composite attributes.

Composite attributes can form a hierarchy; for example, StreetAddress can be subdivided into three simple attributes, Number, Street, and ApartmentNumber, as shown in Figure .The value of a composite attribute is the concatenation of the values of its constituent simple attributes.

2) *simpleor atomic attributes :*

Attributes that are not divisible are called **simple** or **atomic attributes.**

EX: Number, street are simple attributes

3) **Single-valued**

Most attributes have a single value for a particular entity; such attributes are called **single-valued.**

EX: Age is a single-valued attribute of person.

4) *Multivalued Attributes*

In some cases an attribute can have a set of values for the same entity—for example, a Colors attribute fora car, or a CollegeDegrees attribute for a person. Cars with one color have a single value, whereas two-tone cars have two values for Colors. A multivalued attribute may have lower and upper bounds on the number of values allowed for each individual entity.

Ex:The Colors attribute of a car may have between one and three values, if we assume that a car can haveat most three colors.

5) *Stored Versus Derived Attributes*

In some cases two (or more) attribute values are related—for example, the Age and BirthDate attributes of a person. For a particular person entity, the value of Age can be determined from the current (today's) date and the value of that person's BirthDate. The Age attribute is hence called a **derived attribute** and is said to be **derivable from** the BirthDate attribute, which is called a **stored attribute.**

6) Null Values

In some cases a particular entity may not have an applicable value for an attribute. For example, the ApartmentNumber attribute of an address applies only to addresses that are in apartment buildings and not to other types of residences, such as single-family homes. For such situations, a special value called **null** is created. An address of a single-family home would have null for its ApartmentNumber attribute. Null can also be used if we do not know the value of an attribute for a particular entity—for example, if we do not know the home phone of "John Smith" .The meaning of the former type of null is *not applicable,* whereas the meaning of the latter is *unknown.*

The unknown category of null can be further classified into two cases. The first case arises when it is known that the attribute value exists but is *missing*—for example, if the Height attribute of a person is listed as null.

The second case arises when it is *not known* whether the attribute value exists—for example, if the HomePhone attribute of a person is null.

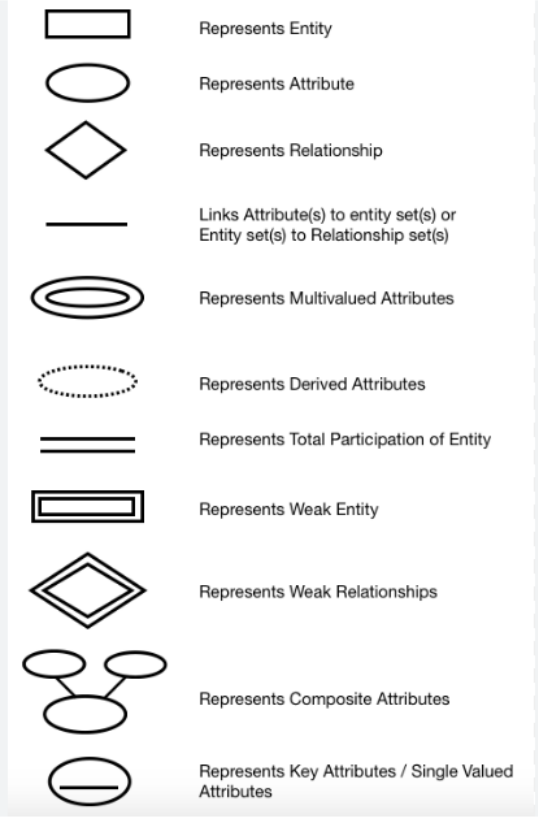
7) Complex Attributes

Notice that composite and multivalued attributes can be nested in an arbitrary way. We can represent arbitrary nesting by grouping components of a composite attribute between parentheses ( ) and separating the components with commas, and by displaying multivalued attributes between braces {}. Such attributes are called **complex attributes.**

1) For example, PreviousDegrees of a STUDENT is acomposite multi-valued attribute denoted by

{PreviousDegrees (College, Year, Degree, Field)}

2) If a person can have more than one residence and each residence can have multiple phones, an attribute AddressPhone for a PERSON entity type can be specified as complex attribute.



6. Explain SQL data definition and data types in brief and explain different CREATE and ALTER command.

Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.

```
CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);
CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);
DROP TABLE ;
DROP TABLE EMPLOYEE;
ALTER TABLE table_name ADD column_name COLUMN-definition;
ALTER TABLE MODIFY(COLUMN DEFINITION....);

ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));

ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));
```

- **SQL Data Definition:**
- The set of relations in a database must be specified to the system by means of a data definition language (DDL).
- It allows specification of not only a set of relations, but also information about each relation, including:
- The schema for each relation.
- The types of values associated with each attribute.
- The integrity constraints.
- The set of indices to be maintained for each relation.
- The security and authorization information for each relation.
- The physical storage structure of each relation on disk
- Basic Types: The SQL standard supports a variety of built-in types, including:
  - Char(n): A fixed-length character string with user-specified length n
  - varchar(n): A variable-length character string with user-specified maximum length n
  - int: An integer (a finite subset of the integers that is machine dependent)
  - smallint: A small integer (a machine-dependent subset of the integer type)
  - numeric(p, d): A fixed-point number with user-specified precision.
  - real, double precision: Floating-point and double-precision floating-point numbers with machine-dependent precision

- float(n): A floating-point number, with precision of at least n digits
- The general form of the create table command is:
- create table r (A1 D1, A2 D2, . . . , An Dn, <integrity-constraint1>, . . . , <integrity-constraintk);
- **Data Types:**
- the SQL standard supports several data types relating to dates and times:
- date: A calendar date containing a (four-digit) year, month, and day of the month
- time: The time of day, in hours, minutes, and seconds
- timestamp: A combination of date and time
- Default Values
- SQL allows a default value to be specified for an attribute as illustrated by the following create table statement:
- Large-Object Types
- SQL provides large-object (LOB) data types for character data (clob) and binary data (blob).
- book_review clob(10KB)
- image blob(10MB)
- movie blob(2GB)
- User-Defined Types
- SQL supports two forms of user-defined data types.

(i) Distinct types

(ii) structured data types: allows the creation of complex data types with nested record structures, arrays, and multi-sets

(iii) Index Creation: An index on an attribute of a relation is a data structure that allows the database system to find those tuples in the relation that have a specified value for that attribute efficiently, without scanning through all the tuples of the relation

7. Create the table named 'employee' with the attributes: empno, empname, dept, designation, salary, doj, place. Write SQL statements for the following

i. Display all the fields of employee table.
ii. Display details of employee number and their salary.
iii. Display average salary of all employees.
iv. Display distinct name of employees in descending order.
v. Count number of employees.
vi. Display the employees whose name contains second and third letters as 'um'.
vii. Display salary of employee which is greater than 120000.
viii. Display details of employee whose name is 'Amit' and salary greater than 50000

Solution:

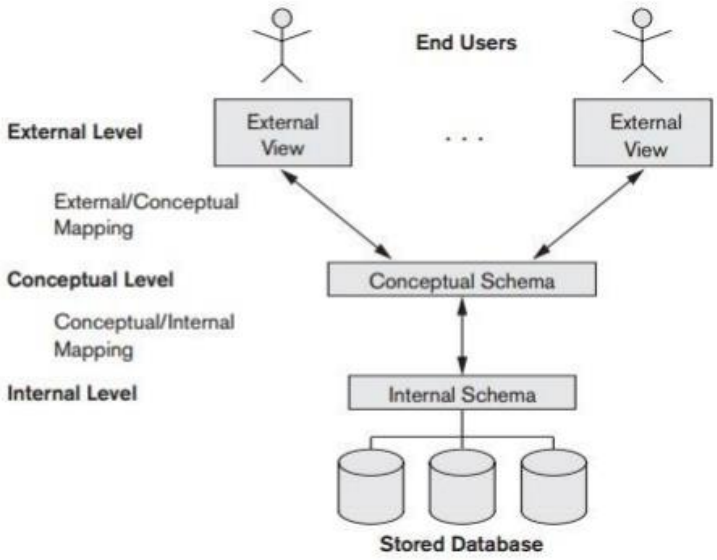
Sql> Select *from employee;
Sql>Select empno,salary from employee;
Sql>select avg(salary)from employee;
Sql>select distinct empname from employee order by empname desc;
Sql>select empname from employee where empname like '_um%';
Sql> select count * from employee;
Sql> select * from employee where salary > 120000
Sql>select *from employee wehre name="Amit" and salary >50000

8. Define data independence and mapping. Explain the three-schema architecture with diagram.

The Three-Schema Architecture

The goal of the three-schema architecture, illustrated in Figure 2.2, is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:

The **internal level** has an **internal schema**, which describes the physical stor-age 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.



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.

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. As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level data model.

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 and explicitly, but support the three-schema architecture to some extent. Some older DBMSs may include physical-level details in the conceptual schema. The three-level ANSI architecture has an important place in database technology development because it clearly separates the users' external level, the database's con-ceptual level, and the internal storage level for designing a database. It is very much applicable in the design of DBMSs, even today. In most DBMSs that support user views, external schemas are specified in the same data model that describes the conceptual-level information (for example, a relational DBMS like Oracle uses SQL for this). Some DBMSs allow different data models to be used at the conceptual and external levels. An example is Universal Data Base (UDB), a DBMS from IBM, which uses the relational model to describe the conceptual schema, but may use an object-oriented model to describe an external schema.

Notice that the three schemas are only *descriptions* of data; the stored data that *actually* exists is at the physical level only. In a DBMS based on the three-schema architecture, each user group refers 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. Even in such systems, however, a certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

2. Data Independence

The three-schema architecture can be used to further 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. We can define two types of data independence:

Logical data independence is the capacity to change the conceptual schema without having to change external schemas 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. For example, the external schema of Figure 1.5(a) should not be affected by changing the GRADE_REPORT file (or record type) shown in Figure 1.2 into the one shown in Figure 1.6(a). Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence. After the conceptual schema undergoes a logical reorganization, application pro-grams that reference the external schema constructs must work as before.

Changes to constraints can be applied to the conceptual schema without affecting the external schemas or application programs.

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 were 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. For example, providing an access path to improve retrieval speed of section records (Figure 1.2) by semester and year should not require a query such as *list all sections offered in fall 2008* to be changed, although the query would be executed more efficiently by the DBMS by utilizing the new access path.

Generally, physical data independence exists in most databases and file environments where physical details such as the exact location of data on disk, and hard-ware details of storage encoding, placement, compression, splitting, merging of records, and so on are hidden from the user. Applications remain unaware of these details. On the other hand, logical data independence is harder to achieve because it allows structural and constraint changes without affecting application programs—a much stricter requirement.

9. List and explain the data types that are allowed for SQL attributes with example. How char data type differs from varchar?

- ✓ The basic data types available for attributes include numeric, character string, bit string, Boolean,date, and time.

Numeric data types

- Integer numbers of various sizes like INTEGER or INT, and SMALLINT

- Floating-point (real) numbers of various precision like FLOAT or REAL, and DOUBLE PRECISION
- Formatted numbers using DECIMAL(i, j)—or DEC(i, j) or NUMERIC(i, j)—where i, the precision, is the total number of decimal digits and j, the scale, is the number of digits after the decimal point.

Character-string datatypes

- Fixed length—CHAR(n) or CHARACTER(n), where n is the number of character
- Varying length— VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n), where n is the maximum number of characters.

Bit-string data types

- Fixed length n—BIT(n)—or varying length— BIT VARYING(n), where n is the maximum number of bits.
- Another variable-length bitstring data type called **BINARY LARGE OBJECT or BLOB** is also availableto specify columns that have large binary values, such as images.

Boolean data types

- Values of TRUE or FALSE.
- In SQL, because of the presence of NULL values, a three-valued logic is used, so a third possible value for a Boolean data type is UNKNOWN.

DATE data types

- Date has ten positions, and its components are YEAR, MONTH, and DAY in the form YYYY-MM-DD.
- The TIME data type has at least eight positions, with the components HOUR, MINUTE, and SECOND in the form HH:MM:SS.

Timestamp data type (TIMESTAMP)

- Includes the DATE and TIME fields, plus a minimum of six positions for decimal fractions of seconds and an optional WITH TIME ZONE qualifier.

Interval

- Another data type related to DATE, TIME, and TIMESTAMP is the INTERVAL data type.
- This specifies an interval—a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp.

VARCHAR is variable length, while CHAR is fixed length.

CHAR is a fixed length string data type, so any remaining space in the field is padded with blanks. CHAR takes up 1 byte per character. So, a CHAR(100) field (or variable) takes up 100 bytes on disk, regardless of the string it holds.

VARCHAR is a variable length string data type, so it holds only the characters you assign to it. VARCHAR takes up 1 byte per character, + 2 bytes to hold length information. For example, if you set a VARCHAR(100) data type = ‘Jen’, then it would take up 3 bytes (for J, E, and N) plus 2 bytes, or 5 bytes in all.

10. With a neat diagram explain the phases of database design.

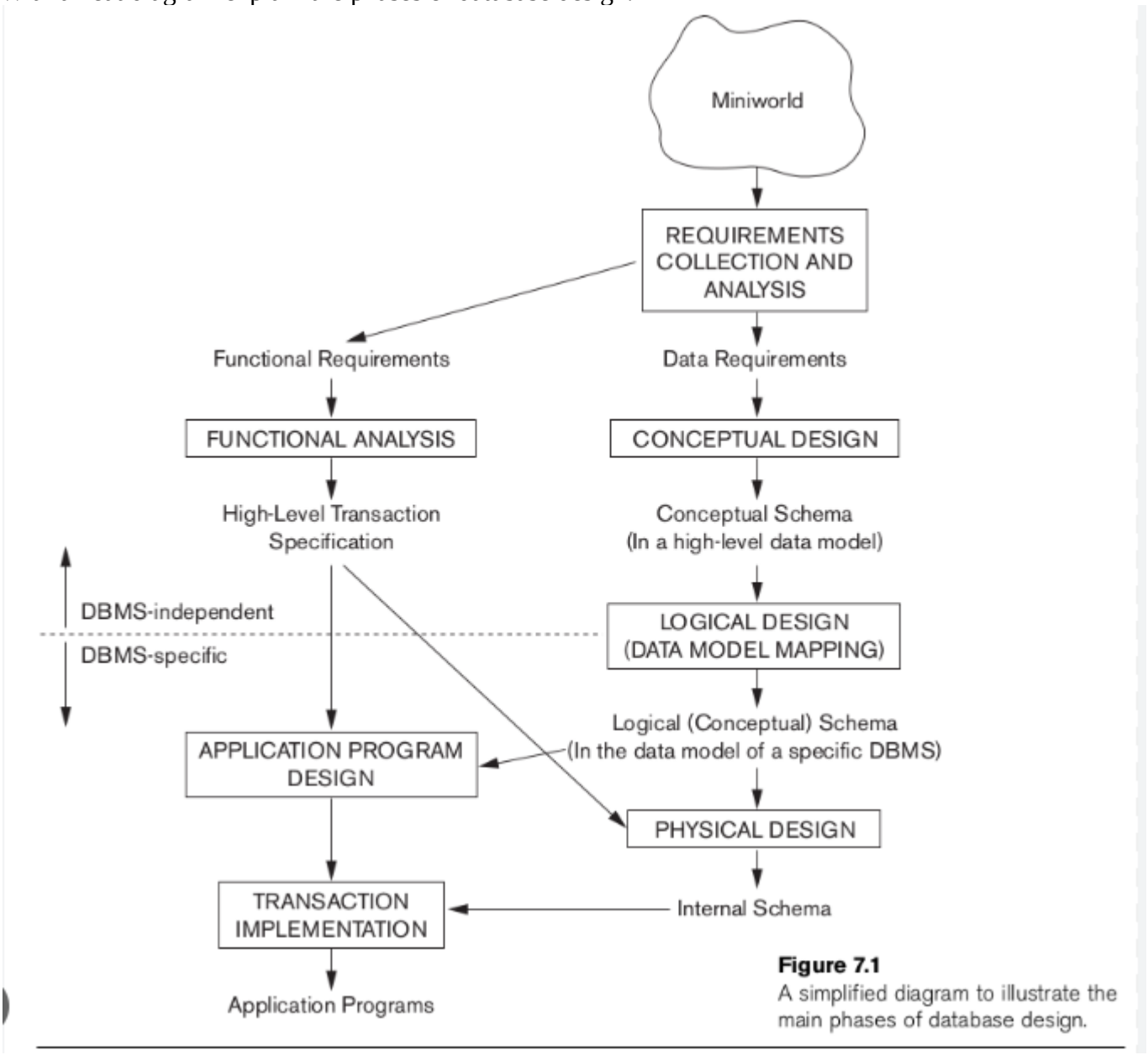


Figure 7.1
A simplified diagram to illustrate the main phases of database design.

Above Figure shows a simplified description of the database design process. The first step shown is **requirements collection and analysis.** During this step, the database designers interview prospective database users to understand and document their **data requirements.** The result of this step is a concisely written set of users' requirements. These requirements should be specified in as detailed and complete a form as possible. In parallel with specifying the data requirements, it is useful to specify the known **functional requirements** of the application. These consist of the user-defined **operations** (or **transactions**) that will be applied to the database, and they include both retrievals and updates.

Once all the requirements have been collected and analyzed, the next step is to create a **conceptual schema** for the database, using a high-level conceptual data model. This step is called **conceptual design.** The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints; these are expressed using the concepts provided by the high-level data model. Because these concepts do not include implementation details, they are usually easier to understand and can be used to communicate with nontechnical users..

During or after the conceptual schema design, the basic data model operations can be used to specify the high-level user operations identified during functional analysis. This also serves to confirm that the conceptual schema meets all the identified functional requirements. Modifications to the conceptual schema

can be introducedif some functional requirements cannot be specified in the initial schema.

The next step in database design is the actual implementation of the database, using a commercial DBMS. Most current commercial DBMSs use an implementation data model—such as the relational or the object database model—so the conceptual schema is transformed from the high-level data model into the implementation data model. This step is called **logical design** or **data model mapping,** and its result is a database schema in the implementation data model of the DBMS.

Finally, the last step is the **physical design** phase, during which the internal storage structures, access paths, and file organizations for the database files are specified. In parallel with these activities, application programs are designed and implemented as database transactions corresponding to the high-level transaction specifications