DATABASE MANAGEMENT SYSTEMS

Subject Code: 10CS54 I.A. Marks: 25 Hours/Week: 04
Exam Hours: 03 Total Hours: 52 Exam Marks: 100

PART - A

UNIT – 1 **6 Hours**

Introduction: Introduction; An example; Characteristics of Database approach; Actors on the screen; Workers behind the scene; Advantages of using DBMS approach; A brief history of database applications; when not to use a DBMS. Data models, schemas and instances; Three-schema architecture and data independence; Database languages and interfaces; the database system environment; Centralized and client-server architectures; Classification of Database Management systems.

UNIT – 2 **6 Hours**

Entity-Relationship Model: Using High-Level Conceptual Data Models for Database Design; An Example Database Application; Entity Types, Entity Sets, Attributes and Keys; Relationship types, Relationship Sets, Roles and Structural Constraints; Weak Entity Types; Refining the ER Design; ER Diagrams, Naming Conventions and Design Issues; Relationship types of degree higher than two.

UNIT – 3 **8 Hours**

Relational Model and Relational Algebra: Relational Model Concepts; Relational Model Constraints and Relational Database Schemas; Update Operations, Transactions and dealing with constraint violations; Unary Relational Operations: SELECT and PROJECT; Relational Algebra Operations from Set Theory; Binary Relational Operations: JOIN and DIVISION; Additional Relational Operations; Examples of Queries in Relational Algebra; Relational Database Design Using ER- to-Relational Mapping.

UNIT – 4 **6 Hours**

SQL – 1: SQL Data Definition and Data Types; Specifying basic constraints in SQL; Schema change statements in SQL; Basic queries in SQL; More complex SQL Queries.

PART - B

UNIT – 5 **6 Hours**

SQL – 2 : Insert, Delete and Update statements in SQL; Specifying constraints as Assertion and Trigger; Views (Virtual Tables) in SQL; Additional features of SQL; Database programming issues and techniques; Embedded SQL, Dynamic SQL; Database stored procedures and SQL / PSM.

UNIT – 6 **6 Hours**

Database Design – 1: Informal Design Guidelines for Relation Schemas; Functional ependencies; Normal Forms Based on Primary Keys; General Definitions of Second and Third Normal Forms; Boyce-Codd Normal Form

UNIT – 7 **6 Hours**

Database Design -2: Properties of Relational Decompositions; Algorithms for Relational Database Schema Design; Multivalued Dependencies and Fourth Normal Form; Join Dependencies and Fifth Normal Form; Inclusion Dependencies; Other Dependencies and Normal Forms

UNIT – 8 **8 Hours**

Transaction Management: The ACID Properties; Transactions and Schedules; Concurrent Execution of Transactions; Lock- Based Concurrency Control; Performance of locking; Transaction support in SQL; Introduction to crash recovery; 2PL, Serializability and Recoverability; Lock Management; Introduction to ARIES; The log; Other recovery-related structures; The write-ahead log protocol; Check pointing; Recovering from a System Crash; Media Recovery; Other approaches and interaction with concurrency control.

Text Books:

1. Elmasri and Navathe: Fundamentals of Database Systems,5th Edition, Pearson Education, 2007.(Chapters 1, 2, 3 except 3.8, 5, 6.1 to 6.5, 7.1, 8, 9.1, 9.2 except SQLJ, 9.4, 10)
2. Raghu Ramakrishnan and Johannes Gehrke: Database Management Systems, 3rd Edition, McGraw-Hill, 2003. (Chapters 16, 17.1, 17.2, 18)

Reference Books:

1. Silberschatz, Korth and Sudharshan: Data base System Concepts, 6th Edition, Mc-GrawHill, 2010.
2. C.J. Date, A. Kannan, S. Swamynatham: An Introduction to Database Systems, 8th Edition, Pearson Education, 2006

INDEX SHEET

Contents **Page No**

UNIT 1

Databases and Database Users

1.1 Introduction

1.2: An Example:

1.3: Characteristics of the Database Approach:

1.4: Actors on the Scene

1.5: Workers Behind the Scene

1.6: Capabilities/Advantages of DBMS's

1.7: A Brief History of Database Applications

1.8: When Not to Use a DBMS

Unit 1 Databases and Database Users

1.1 Introduction

Importance: Database systems have become an essential component of life in modern society, in that many frequently occurring events trigger the accessing of at least one database: bibliographic library searches, bank transactions, hotel/airline reservations, grocery store purchases, online (Web) purchases, etc., etc.

Traditional vs. more recent applications of databases:

The applications mentioned above are all "traditional" ones for which the use of rigidly-structured textual and numeric data suffices. Recent advances have led to the application of database technology to a wider class of data. Examples include **multimedia** databases (involving pictures, video clips, and sound messages) and **geographic** databases (involving maps, satellite images).

Also, database search techniques are applied by some WWW search engines.

Definitions

The term **database** is often used, rather loosely, to refer to just about any collection of related data. E&N say that, in addition to being a collection of related data, a database must have the following properties:

- It represents some aspect of the real (or an imagined) world, called the **miniworld** or **universe of discourse**. Changes to the miniworld are reflected in the database. Imagine, for example, a UNIVERSITY miniworld concerned with students, courses, course sections, grades, and course prerequisites.
- It is a logically coherent collection of data, to which some meaning can be attached. (Logical coherency requires, in part, that the database not be self-contradictory.)
- It has a purpose: there is an intended group of users and some preconceived applications that the users are interested in employing.

To summarize: a database has some source (i.e., the miniworld) from which data are derived, some degree of interaction with events in the represented miniworld (at least insofar as the data is updated when the state of the miniworld changes), and an audience that is interested in using it.

An Aside: data vs. information vs. knowledge: Data is the representation of "facts" or "observations" whereas information refers to the meaning thereof (according to some interpretation). Knowledge, on the other hand, refers to the ability to use information to achieve intended ends.

Computerized vs. manual: Not surprisingly (this being a CS course), our concern will be with computerized database systems, as opposed to manual ones, such as the card catalog-based

systems that were used in libraries in ancient times (i.e., before the year 2000). (Some authors wouldn't even recognize a non-computerized collection of data as a database, but E&N do.)

Size/Complexity: Databases run the range from being small/simple (e.g., one person's recipe database) to being huge/complex (e.g., Amazon's database that keeps track of all its products, customers, and suppliers).

Definition: A **database management system** (DBMS) is a collection of programs enabling users to create and maintain a database.

More specifically, a DBMS is a general purpose software system facilitating each of the following (with respect to a database):

- **definition:** specifying data types (and other constraints to which the data must conform) and data organization
- **construction:** the process of storing the data on some medium (e.g., magnetic disk) that is controlled by the DBMS
- **manipulation:** querying, updating, report generation
- **sharing:** allowing multiple users and programs to access the database "simultaneously"
- **system protection:** preventing database from becoming corrupted when hardware or software failures occur
- **security protection:** preventing unauthorized or malicious access to database.

Given all its responsibilities, it is not surprising that a typical DBMS is a complex piece of software.

A database together with the DBMS software is referred to as a **database system**. (See Figure 1.1, page 7.)

1.2: An Example:

UNIVERSITY database in Figure 1.2. Notice that it is relational!

Among the main ideas illustrated in this example is that each file/relation/table has a set of named fields/attributes/columns, each of which is specified to be of some data type. (In addition to a data type, we might put further restrictions upon a field, e.g., GRADE_REPORT must have a value from the set {'A', 'B', ..., 'F'}.)

The idea is that, of course, each table will be populated with data in the form of records/tuples/rows, each of which represents some entity (in the miniworld) or some relationship between entities.

For example, each record in the **STUDENT** table represents a —surprise!— student. Similarly for the **COURSE** and **SECTION** tables.

On the other hand, each record in **GRADE_REPORT** represents a relationship between a student and a section of a course. And each record in **PREREQUISITE** represents a relationship between two courses.

Database manipulation involves querying and updating.

Examples of (informal) queries:

- Retrieve the transcript(s) of student(s) named 'Smith'.
- List the names of students who were enrolled in a section of the 'Database' course in Spring 2006, as well as their grades in that course section.
- List all prerequisites of the 'Database' course.

Examples of (informal) updates:

- Change the CLASS value of 'Smith' to sophomore (i.e., 2).
- Insert a record for a section of 'File Processing' for this semester.
- Remove from the prerequisites of course 'CMPS 340' the course 'CMPS 144'.

Of course, a query/update must be conveyed to the DBMS in a precise way (via the query language of the DBMS) in order to be processed.

As with software in general, developing a new database (or a new application for an existing database) proceeds in phases, including **requirements analysis** and various levels of **design** (conceptual (e.g., Entity-Relationship Modeling), logical (e.g., relational), and physical (file structures)).

1.3: Characteristics of the Database Approach:

Database approach vs. File Processing approach: Consider an organization/enterprise that is organized as a collection of departments/offices. Each department has certain data processing "needs", many of which are unique to it. In the **file processing approach**, each department would control a collection of relevant data files and software applications to manipulate that data.

For example, a university's Registrar's Office would maintain data (and programs) relevant to student grades and course enrollments. The Bursar's Office would maintain data (and programs) pertaining to fees owed by students for tuition, room and board, etc. (Most likely, the people in these offices would not be in direct possession of their data and programs, but rather the university's Information Technology Department would be responsible for providing services such as data storage, report generation, and programming.)

One result of this approach is, typically, **data redundancy**, which not only wastes storage space but also makes it more difficult to keep changing data items consistent with one another, as a change to one copy of a data item must be made to all of them (called **duplication-of-effort**). **Inconsistency** results when one (or more) copies of a datum are changed but not others. (E.g., If

you change your address, informing the Registrar's Office should suffice to ensure that your grades are sent to the right place, but does not guarantee that your next bill will be, as the copy of your address "owned" by the Bursar's Office might not have been changed.)

In the **database approach**, a single repository of data is maintained that is used by all the departments in the organization. (Note that "single repository" is used in the logical sense. In physical terms, the data may be distributed among various sites, and possibly mirrored.)

Main Characteristics of database approach:

1. **Self-Description:** A database system includes —in addition to the data stored that is of relevance to the organization— a complete definition/description of the database's structure and constraints. This **meta-data** (i.e., data about data) is stored in the so-called **system catalog**, which contains a description of the structure of each file, the type and storage format of each field, and the various constraints on the data (i.e., conditions that the data must satisfy).

   See Figures 1.1 and 1.3.

   The system catalog is used not only by users (e.g., who need to know the names of tables and attributes, and sometimes data type information and other things), but also by the DBMS software, which certainly needs to "know" how the data is structured/organized in order to interpret it in a manner consistent with that structure. Recall that a DBMS is general purpose, as opposed to being a specific database application. Hence, the structure of the data cannot be "hard-coded" in its programs (such as is the case in typical file processing approaches), but rather must be treated as a "parameter" in some sense.

2. **Insulation between Programs and Data; Data Abstraction:**

   Program-Data Independence: In traditional file processing, the structure of the data files accessed by an application is "hard-coded" in its source code. (E.g., Consider a file descriptor in a COBOL program: it gives a detailed description of the layout of the records in a file by describing, for each field, how many bytes it occupies.)

   If, for some reason, we decide to change the structure of the data (e.g., by adding the first two digits to the YEAR field, in order to make the program Y2K compliant!), **every** application in which a description of that file's structure is hard-coded must be changed!

   In contrast, DBMS access programs, in most cases, do not require such changes, because the structure of the data is described (in the system catalog) separately from the programs that access it and those programs consult the catalog in order to ascertain the structure of the data (i.e., providing a means by which to determine boundaries between records and between fields within records) so that they interpret that data properly.

   See Figure 1.4.

In other words, the DBMS provides a conceptual or logical view of the data to application programs, so that the underlying implementation may be changed without the programs being modified. (This is referred to as program-data independence.)

Also, which access paths (e.g., indexes) exist are listed in the catalog, helping the DBMS to determine the most efficient way to search for items in response to a q uery.

Data Abstraction:

- A **data model** is used to hide storage details and present the users with a conceptual view of the database.
- Programs refer to the data model constructs rather than data storage details

Note: In fairness to COBOL, it should be pointed out that it has a COPY feature that allows different application programs to make use of the same file descriptor stored in a "library". This provides some degree of program-data independence, but not nearly as much as a good DBMS does. End of note.

Example by which to illustrate this concept: Suppose that you are given the task of developing a program that displays the contents of a particular data file. Specifically, each record should be displayed as follows:

```
Record #i:
  value of first field
  value of second field
  ...
  ...
  value of last field
```

To keep things very simple, suppose that the file in question has fixed-length records of 57 bytes with six fixed-length fields of lengths 12, 4, 17, 2, 15, and 7 bytes, respectively, all of which are ASCII strings. Developing such a program would not be difficult. However, the obvious solution would be tailored specifically for a file having the particular structure described here and would be of no use for a file with a different structure.

Now suppose that the problem is generalized to say that the program you are to develop must be able to display any file having fixed-length records with fixed-length fields that are ASCII strings. Impossible, you say? Well, yes, unless the program has the ability to access a description of the file's structure (i.e., lengths of its records and the fields therein), in which case the problem is not hard at all. This illustrates the power of metadata, i.e., data describing other data.

3. **Multiple Views of Data:** Different users (e.g., in different departments of an organization) have different "views" or perspectives on the database. For example, from the point of view of a Bursar's Office employee, student data does not include anything about which courses were taken or which grades were earned. (This is an example of a **subset** view.)

   As another example, a Registrar's Office employee might think that GPA is a field of data in each student's record. In reality, the underlying database might calculate that value each time it is needed. This is called **virtual** (or **derived**) data.

   A view designed for an academic advisor might give the appearance that the data is structured to point out the prerequisites of each course.

   (See Figure 1.5, page 14.)

   A good DBMS has facilities for defining multiple views. This is not only convenient for users, but also addresses security issues of data access. (E.g., The Registrar's Office view should not provide any means to access financial data.)

4. **Data Sharing and Multi-user Transaction Processing:** As you learned about (or will) in the OS course, the simultaneous access of computer resources by multiple users/processes is a major source of complexity. The same is true for multi-user DBMS's.

   Arising from this is the need for **concurrency control**, which is supposed to ensure that several users trying to update the same data do so in a "controlled" manner so that the results of the updates are as though they were done in some sequential order (rather than interleaved, which could result in data being incorrect).

   This gives rise to the concept of a **transaction**, which is a process that makes one or more accesses to a database and which must have the appearance of executing in isolation from all other transactions (even ones that access the same data at the "same time") and of being atomic (in the sense that, if the system crashes in the middle of its execution, the database contents must be as though it did not execute at all).

   Applications such as **airline reservation systems** are known as **online transaction processing** applications.

1.4: Actors on the Scene

These apply to "large" databases, not "personal" databases that are defined, constructed, and used by a single person via, say, Microsoft Access.

- Users may be divided into
  - Those who actually use and control the database content, and those who design, develop and maintain database applications (called "Actors on the Scene"), and
  - Those who design and develop the DBMS software and related tools, and the computer systems operators (called "Workers Behind the Scene").

1. **Database Administrator** (DBA): This is the chief administrator, who oversees and manages the database system (including the data and software). Duties include authorizing users to access the database, coordinating/monitoring its use, acquiring hardware/software for upgrades, etc. In large organizations, the DBA might have a support staff.
2. **Database Designers**: They are responsible for identifying the data to be stored and for choosing an appropriate way to organize it. They also define **views** for different categories of users. The final design must be able to support the requirements of all the user sub-groups.
3. **End Users**: These are persons who access the database for **querying**, **updating**, and **report generation**. They are main reason for database's existence!
   - **Casual end users:** use database occasionally, needing different information each time; use query language to specify their requests; typically middle- or high-level managers.
   - **Naive/Parametric end users**: Typically the biggest group of users; frequently query/update the database using standard **canned transactions** that have been carefully programmed and tested in advance. Examples:
     - bank tellers check account balances, post withdrawals/deposits
     - reservation clerks for airlines, hotels, etc., check availability of seats/rooms and make reservations.
     - shipping clerks (e.g., at UPS) who use buttons, bar code scanners, etc., to update status of in-transit packages.
   - **Sophisticated end users**: engineers, scientists, business analysts who implement their own applications to meet their complex needs.
   - **Stand-alone users**: Use "personal" databases, possibly employing a special-purpose (e.g., financial) software package. Mostly maintain personal databases using ready-to-use packaged applications.
   - An example is a tax program user that creates its own internal database.
   - Another example is maintaining an address book
4. **System Analysts, Application Programmers, Software Engineers**:
   - **System Analysts**: determine needs of end users, especially naive and parametric users, and develop specifications for canned transactions that meet these needs.
   - **Application Programmers**: Implement, test, document, and maintain programs that satisfy the specifications mentioned above.

1.5: Workers Behind the Scene

- **DBMS system designers/implementors:** provide the DBMS software that is at the foundation of all this!

- **Tool developers**: design and implement software tools facilitating database system design, performance monitoring, creation of graphical user interfaces, prototyping, ets.
- **Operators and maintenance personnel**: responsible for the day-to-day operation of the system.

1.6: Capabilities/Advantages of DBMS's

1. **Controlling Redundancy:** Data redundancy (such as tends to occur in the "file processing" approach) leads to **wasted storage space**, **duplication of effort** (when multiple copies of a datum need to be updated), and a higher liklihood of the introduction of **inconsistency**.

   On the other hand, redundancy can be used to improve performance of queries. Indexes, for example, are entirely redundant, but help the DBMS in processing queries more quickly.

   Another example of using redundancy to improve performance is to store an "extra" field in order to avoid the need to access other tables (as when doing a JOIN, for example). See Figure 1.6 (page 18): the StudentName and CourseNumber fields need not be there.

   A DBMS should provide the capability to automatically enforce the rule that no inconsistencies are introduced when data is updated. (Figure 1.6 again, in which Student_name does not match Student_number.)

2. **Restricting Unauthorized Access:** A DBMS should provide a **security and authorization subsystem**, which is used for specifying restrictions on user accounts. Common kinds of restrictions are to allow read-only access (no updating), or access only to a subset of the data (e.g., recall the Bursar's and Registrar's office examples from above).
3. **Providing Persistent Storage for Program Objects:** Object-oriented database systems make it easier for complex runtime objects (e.g., lists, trees) to be saved in secondary storage so as to survive beyond program termination and to be retrievable at a later time.
4. **Providing Storage Structures for Efficient Query Processing:** The DBMS maintains indexes (typically in the form of trees and/or hash tables) that are utilized to improve the execution time of queries and updates. (The choice of which indexes to create and maintain is part of physical database design and tuning (see Chapter 16) and is the responsibility of the DBA.

   The **query processing and optimization** module is responsible for choosing an efficient query execution plan for each query submitted to the system. (See Chapter 15.)

5. **Providing Backup and Recovery:** The subsystem having this responsibility ensures that recovery is possible in the case of a system crash during execution of one or more transactions.

6. **Providing Multiple User Interfaces:** For example, query languages for casual users, programming language interfaces for application programmers, forms and/or command codes for parametric users, menu-driven interfaces for stand-alone users.
7. **Representing Complex Relationships Among Data:** A DBMS should have the capability to represent such relationships and to retrieve related data quickly.
8. **Enforcing Integrity Constraints:** Most database applications are such that the semantics (i.e., meaning) of the data require that it satisfy certain restrictions in order to make sense. Perhaps the most fundamental constraint on a data item is its data type, which specifies the universe of values from which its value may be drawn. (E.g., a Grade field could be defined to be of type Grade_Type, which, say, we have defined as including precisely the values in the set { "A", "A-", "B+", ..., "F" }.

   Another kind of constraint is referential integrity, which says that if the database includes an entity that refers to another one, the latter entity must exist in the database. For example, if (R56547, CIL102) is a tuple in the Enrolled_In relation, indicating that a student with ID R56547 is taking a course with ID CIL102, there must be a tuple in the Student relation corresponding to a student with that ID.

9. **Permitting Inferencing and Actions Via Rules:** In a **deductive** database system, one may specify declarative rules that allow the database to infer new data! E.g., Figure out which students are on academic probation. Such capabilities would take the place of application programs that would be used to ascertain such information otherwise.

   Active database systems go one step further by allowing "active rules" that can be used to initiate actions automatically.

1.7: A Brief History of Database Applications

- Early Database Applications:
  - The Hierarchical and Network Models were introduced in mid 1960s and dominated during the seventies.
  - A bulk of the worldwide database processing still occurs using these models.
- Relational Model based Systems:
  - Relational model was originally introduced in 1970, was heavily researched and experimented with in IBM Research and several universities.
  - Object-oriented and emerging applications:

Object-Oriented Database Management Systems (OODBMSs) were introduced in late 1980s and early 1990s to cater to the need of complex data processing in CAD and other applications.

  - Their use has not taken off much.

Many relational DBMSs have incorporated object database concepts, leading to a new category called object-relational DBMSs (ORDBMSs)

Extended relational systems add further capabilities (e.g. for multimedia data, XML, and other data types)

- Relational DBMS Products emerged in the 1980s
- Data on the Web and E-commerce Applications:
- Web contains data in HTML (Hypertext markup language) with links among pages.
- This has given rise to a new set of applications and E-commerce is using new standards like XML (eXtended Markup Language).
- Script programming languages such as PHP and JavaScript allow generation of dynamic Web pages that are partially generated from a database
    - New functionality is being added to DBMSs in the following areas:
    - Scientific Applications
    - XML (eXtensible Markup Language)
    - Image Storage and Management
    - Audio and Video data management
    - Data Warehousing and Data Mining
    - Spatial data management
    - Time Series and Historical Data Management
    - The above gives rise to new research and development in incorporating new data types, complex data structures, new operations and storage and indexing schemes in database systems.
    - Also allow database updates through Web pages

1.8: When Not to Use a DBMS

Main inhibitors (costs) of using a DBMS:

- High initial investment and possible need for additional hardware.
- Overhead for providing generality, security, concurrency control, recovery, and integrity functions.

- When a DBMS may be unnecessary:
    - If the database and applications are simple, well defined, and not expected to change.
    - If there are stringent real-time requirements that may not be met because of DBMS overhead.
    - If access to data by multiple users is not required.
- When no DBMS may suffice:
    - If the database system is not able to handle the complexity of data because of modeling limitations
    - If the database users need special operations not supported by the DBMS.