Chapter 1 - Databases and Database Users

Introduction

DBMS – A Database is a collection of interrelated data and a Database Management System is a set of programs to use and/or modify this data.

Approaches to Data Management

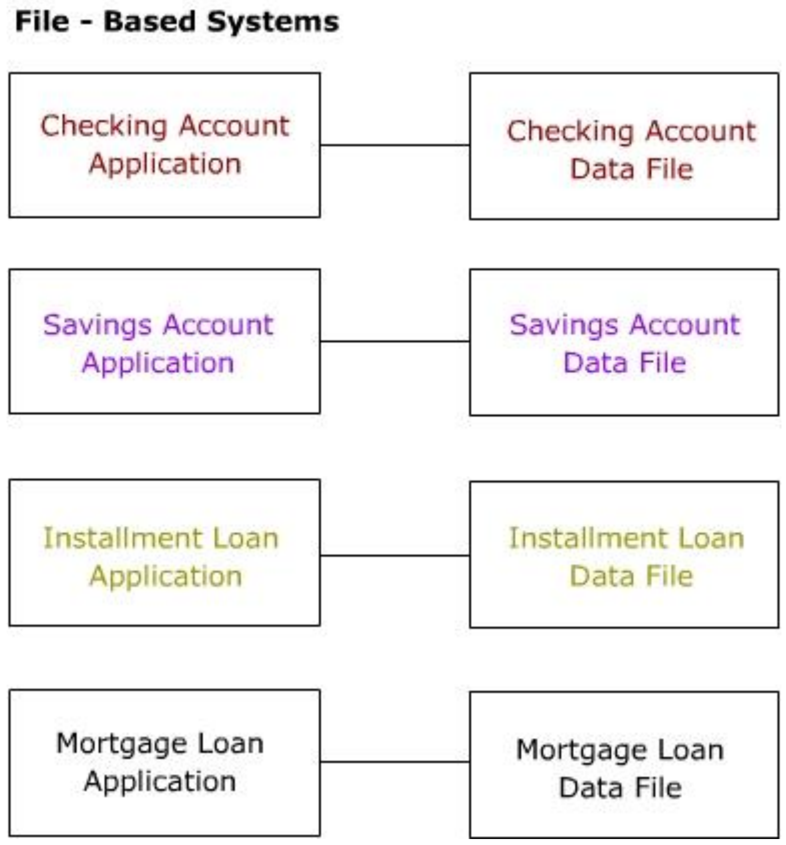
- **File-Based Systems**

  Conventionally, before the Database systems evolved, data in software systems was stored in and represented using flat files.

- **Database Systems**

  Database Systems evolved in the late 1960s to address common issues in applications handling large volumes of data which are also data intensive. Some of these issues could be traced back to the following disadvantages of File-based systems.

Drawbacks of File-Based Systems


File - Based Systems
Checking Account Application
Checking Account Data File
Savings Account Application
Savings Account Data File
Installment Loan Application
Installment Loan Data File
Mortgage Loan Application
Mortgage Loan Data File

As shown in the figure, in a file-based system, different programs in the same application may be interacting with different private data files. There is no system enforcing any standardized control on the organization and structure of these data files.

- **Data Redundancy and Inconsistency**

  Since data resides in different private data files, there are chances of redundancy and resulting inconsistency. For example, in the above example shown, the same customer can have a savings account as well as a mortgage loan. Here the customer details may be duplicated since the programs for the two functions store their corresponding data in two different data files. This gives rise to redundancy in the customer's data. Since the same data is stored in two files, inconsistency arises if a change made in the data in one file is not reflected in the other.

- **Unanticipated Queries**

  In a file-based system, handling sudden/ad-hoc queries can be difficult, since it requires changes in the existing programs.

- **Data Isolation**

  Though data used by different programs in the application may be related, they reside in isolated data files.

- **Concurrent Access Anomali**es

  In large multi-user systems the same file or record may need to be accessed by multiple users simultaneously. Handling this in a file-based systems is difficult.

- **Security Problems**

  In data-intensive applications, security of data is a major concern. Users should be given access only to required data and not the whole database. In a file-based system, this can be handled only by additional programming in each application.

- **Integrity Problems**

  In any application, there will be certain data integrity rules which needs to be maintained. These could be in the form of certain conditions/constraints on the elements of the data records. In the savings bank application, one such integrity rule could be "Customer ID, which is the unique identifier for a customer record, should be non-empty". There can be several such integrity rules. In a file-based system, all these rules need to be explicitly programmed in the application program.

It may be noted that, we are not trying to say that handling the above issues like concurrent access, security, integrity problems, etc., is not possible in a file-based system. The real issue was that, though all these are common issues of concern to any data-intensive application, each application had to handle all these problems on its own. The application programmer needs to bother not only about implementing the application business rules but also about handling these common issues.

Characteristics of the Database Approach

Self-Describing Nature of a Database System:

• A DBMS catalog stores the *description* of the database. The description is called meta-data .

• The catalog is used by the DBMS software and also by database users who need information about the database structure.

• 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.

• The DBMS software must work equally well with any number of database applications—for example, a university database, a banking database, or a company database—as long as the database definition is stored in the catalog.

• 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.

• For example, a PASCAL program may have record structures declared in it; a C++ program may have "struct" or "class" declarations; and a COBOL program has Data Division statements to define its files.

• Whereas file-processing software can access only specific databases, DBMS software can access diverse databases by extracting the database definitions from the catalog and then using these definitions.

Insulation between Programs and Data, and Data Abstraction:

- In a database
  - The structure of data files is stored in the DBMS catalog separately from the access programs.
  - Changes to the structure of data files will not lead to changes to the access programs, this property is called ***program-data independence*.**
- In object_oriented and object-relational databases,
  - Users can define operations on data as part of the database definition.

- User application programs can operate on the data by invoking these operations, regardless of how the operations are implemented. This property is called ***program-operation independence***.
- The characteristic that allows program-data independence and program-operation independence is called ***data abstraction***

Support of Multiple Views of the Data:

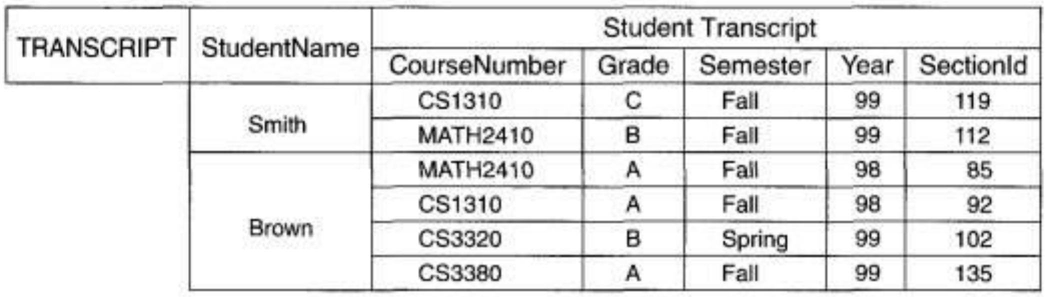
• A database typically has many users, each of whom may require a different perspective or view of the database.

• A multi-user DBMS whose users have a variety of applications must provide facilities for defining multiple views.

• For example, one user of the database of Figure 2 may be interested only in the transcript of each student; the view for this user is shown in Figure (a).

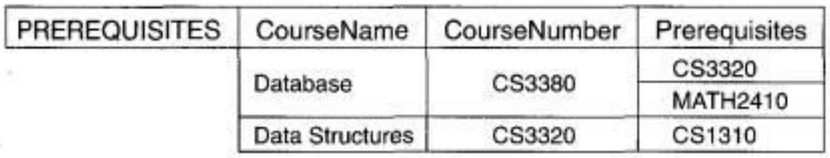
• A second user, who is interested only in checking that students have taken all the prerequisites of each course they register for, may require the view shown in Figure (b).

(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

Sharing of Data and Multi-user Transaction Processing:

• A multi-user DBMS, as its name implies, must allow multiple users to access the database at the same time.

• 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.

• For example, when several reservation clerks try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one clerk at a time for assignment to a passenger.

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.

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.
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.

Workers behind the Scene

- **DBMS system designers/implementers:** 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, etc.
- **Operators and maintenance personnel**: responsible for the day-to-day operation of the system.

Advantages of Using a DBMS

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 likelihood 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 1.5: 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 1.5 again.)

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* 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.

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.

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, particularly, the hierarchical model.
- Relational Model based Systems:
    - Relational model was originally introduced in 1970, was heavily researched and experimented within IBM Research and several universities.
    - Relational DBMS Products emerged in the early 1980s.

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)

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.
    - Also allow database updates through Web pages