Database Management System (MDS 505)

Jagdish Bhatta

Unit-1

Fundamental Concept of DBMS: Data Models

Data Models

- 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. 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.

Data Models

- Data model is a collection of tools for describing;
  - Data
  - Data relationships
  - Data semantics
  - Data constraints

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 on the computer storage media, typically magnetic disks.
- Concepts provided by physical 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**, 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.

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 in 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. The **entity–relationship model** is a popular high-level conceptual data model.

Data Models

- 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. Representational data models represent data by using record structures and hence are sometimes called **record-based data models**.

Data Models

- 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). 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 search structure that makes the search for particular database records efficient, such as indexing or hashing. 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 text, except that it may be organized in a linear, hierarchical (tree-structured), or some other fashion.

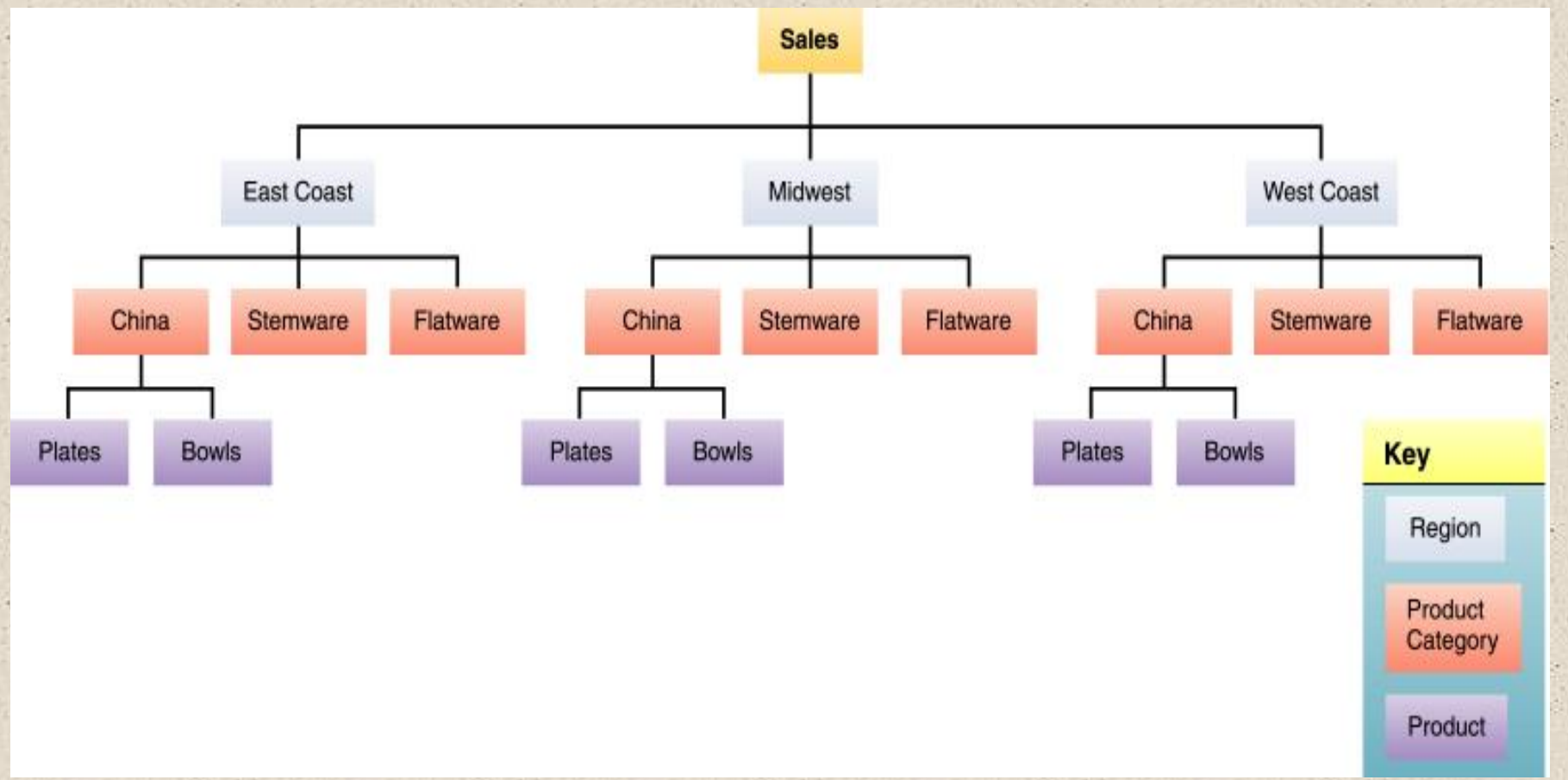
Data Models

- Another class of data models is known as **self-describing data models**. The data storage in systems based on these models combines the description of the data with the data values themselves. In traditional DBMSs, the description (schema) is separated from the data.
- These models include **XML** as well as many of the **key-value stores** and **NOSQL systems** that were recently created for managing big data.

- **Hierarchical Data Model:**
  - A hierarchical database model is a data model in which the data is organized into a tree-like structure.
  - The structure allows representing information using parent/child relationships: each parent can have many children, but each child has only one parent. All attributes of a specific record are listed under an entity type.
  - Hierarchical database model rigidly structures data into an inverted “tree” in which each record contains two elements, a single root or master field, often called a key, and a variable number of subordinate fields.

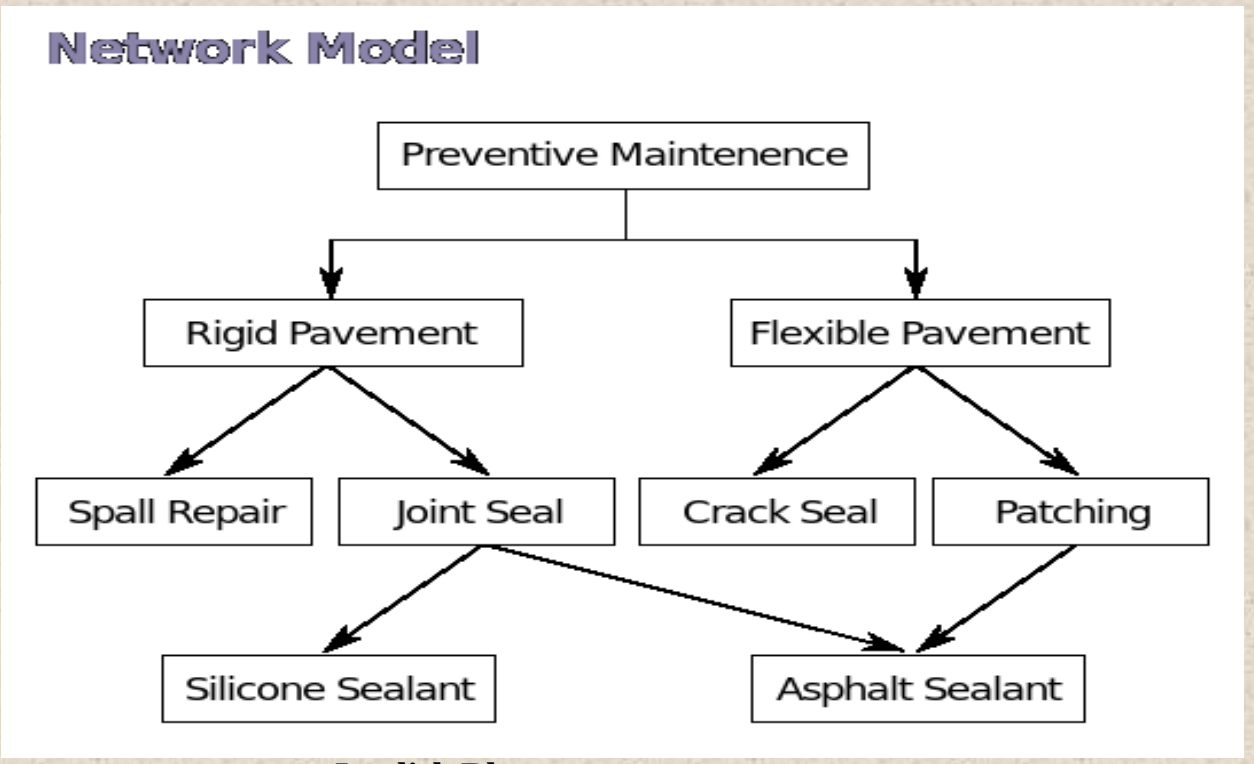
- **Hierarchical Data Model:**
  - The strongest advantage of the hierarchical database approach is the speed and efficiency with which it can be searched for data.
  - The hierarchical model does have problems: Access to data in this model is predefined by the database administrator before the programs that access the data are written. Programmers must follow the hierarchy established by the data structure.
  - It can not handle many to many relationship

- **Hierarchical Data Model:**



- **Network Data Model:**
  - The network model is a database model conceived as a flexible way of representing objects and their relationships.
  - Its distinguishing feature is that the schema, viewed as a graph in which object types are nodes and relationship types are arcs.
  - Unlike hierarchical model, the network model allows each record to have multiple parent and child records, forming a lattice structure. The network model replaces the hierarchical tree with a graph thus allowing more general connections among the nodes. This model was evolved to specially handle non-hierarchical relationships. Now, same data may exist in two different levels.

- **Network Data Model:**
  - Support many-to-many relationship
  - Good processing speed but very complex model to design, implement and maintain



- **Relational Data Model**
  - Relational data model represents the database as a collection of relations where each relation resembles a table of values with rows and columns.
  - A relation may be regarded as a set of **tuples**, also called records. A relation consists of relation schema & relation instance. The **relation schema** specifies relation name & description of tuples (name of attributes, domain). While relation instance corresponds to a table of rows & columns where each row is a tuple & **the column is the attribute**.
  - Major advantages of relational model over the older data models are the simple data representation & the ways complex queries can be expressed easily.

- **Relational Data Model**
  - Consider a relation representing employee record as;

Employee

Eid	Name	Address	Depart_no
011	Ram Sing	Kathmandu	D01
012	Hari Saha	Pokhara	D02

  - The relational model of data permits the database designer to create a consistent, logical representation of information. Consistency is achieved by including declared constraints in the database design, which is usually referred to as the logical schema. The theory includes a process of database normalization whereby a design with certain desirable properties can be selected from a set of logically equivalent alternatives

customer_id	*customer_name*	*customer_street*	*customer_city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account_number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer_id	*account_number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

Fig: A Sample Relational Database

The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model. The relational model is at a lower level of abstraction than the E-R model. Database designs are often carried out in the E-R model, and then translated to the relational model.

- **Entity-Relationship Model:**

- The ***entity-relationship (E-R) model*** is a high level data model based on a perception of a real world that consists of collection of basic objects, called ***entities***, and of ***relationships*** among these entities. An ***entity*** is a thing or object in the real world that is distinguishable from other objects.

- Entities are described in a database by a set of ***attributes***. A ***relationship*** is an association among several entities. The set of all entities of the same type is called an ***entity set*** and the set of all relationships of the same type is called a ***relationship set***.

The overall logical structure of a database can be expressed graphically by an E-R diagram. The components of this diagram are:

- **Rectangles** (represent entity sets)
- **Ellipses** (represent attributes)
- **Diamonds** (represent relationship sets among entity sets)
- **Lines** (link attributes to entity sets and entity sets to relationship sets)

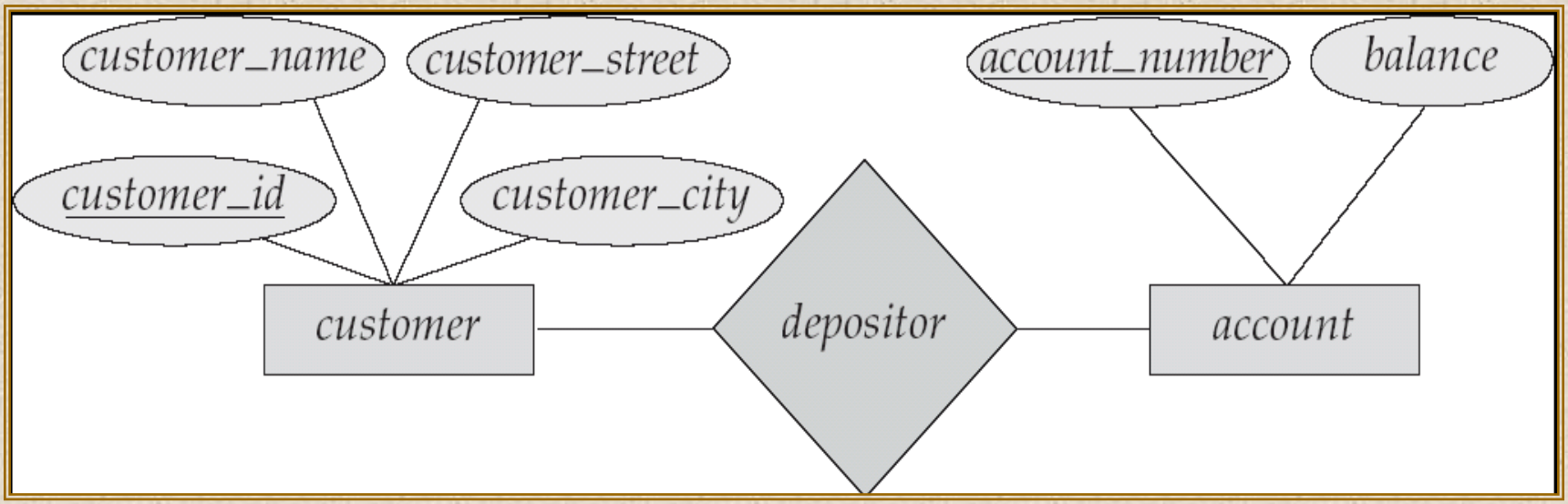
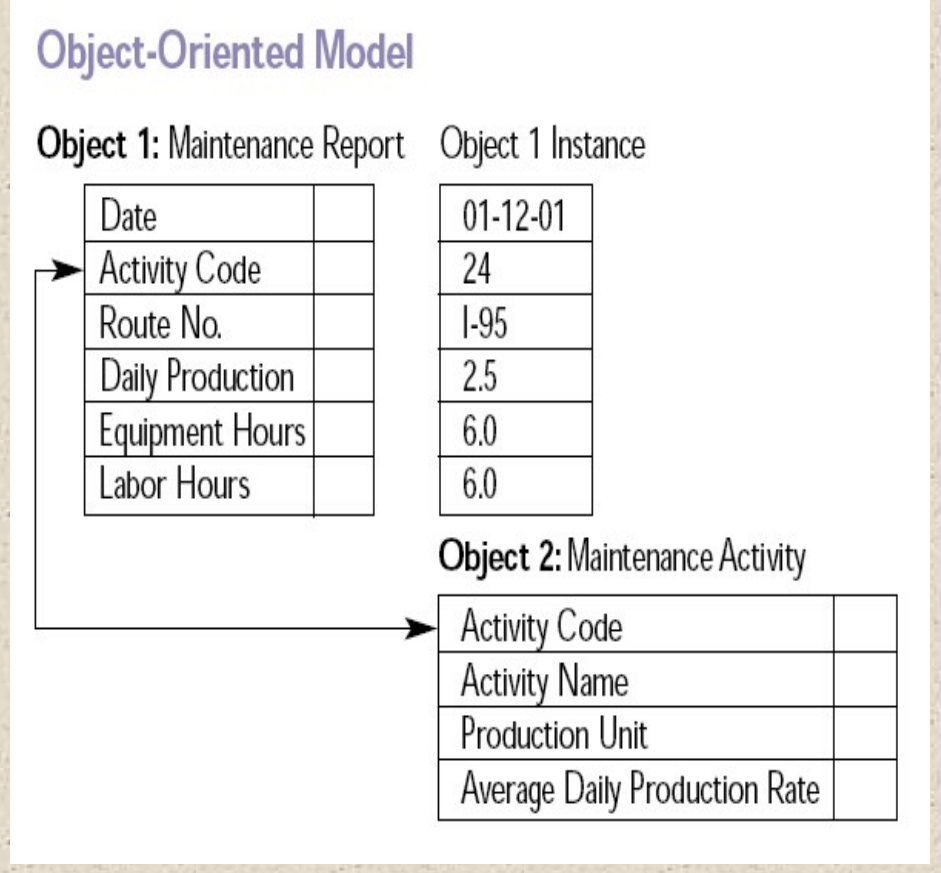


Fig: A Sample E-R Diagram

- **Object Oriented Database:** An object database (also object-oriented data model) is a database model in which information is represented in the form of objects as used in object-oriented programming.
- Data model that adds new object storage capabilities to relational databases.

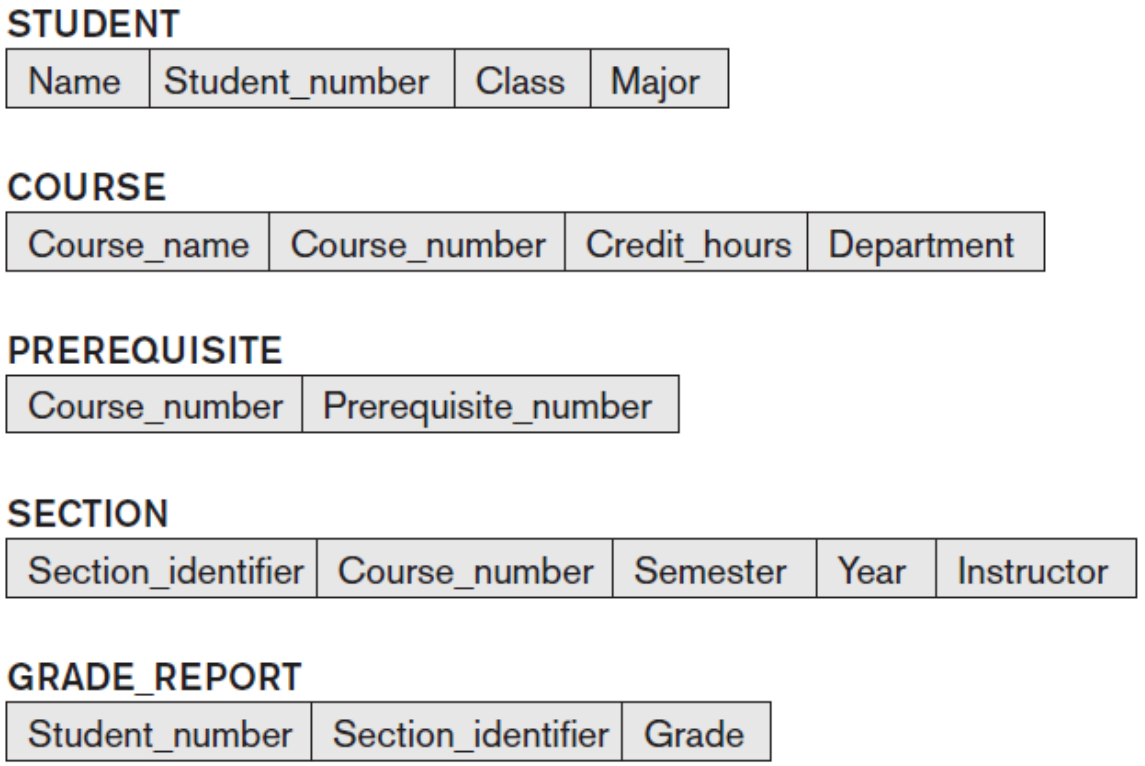


Schemas and Instances

- The description of a database is called the ***database schema,*** which is specified during database design and is not expected to change frequently.
- DB Schema
  - **captures attributes, relationships, constraints on the data**
  - **is independent of any application program**
- Databases change over time as data is inserted and deleted. The collection of data stored in the database at a particular moment of time is called an ***instance*** of the database. Also known as ***DB State***.

Schemas and Instances

Figure 2.1
Schema diagram for the database in Figure 1.2.



Example Of A Simple Database

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

Jagdish Bhatta

Schemas and Instances

- 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**. Other aspects are not specified in the schema diagram; for example, figure 2.1 (in slide number 23) shows neither the data type of each data item nor the relationships among the various files.

Schemas and Instances

- The actual data in a database may change quite frequently. For example, the database shown in Figure 1.2 (in slide number 24) changes every time we add a new student or enter a new grade. 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. Many database states can be constructed to correspond to a particular database schema. Every time we insert or delete a record or change the value of a data item in a record, we change one state of the database into another state.

Schemas and 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. Hence, specifying a correct schema to the DBMS is extremely important and the schema must be designed with utmost care. 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 is called an **extension** of the schema.

Schemas and Instances

- Although, as mentioned earlier, the schema is not supposed to change frequently, it is not uncommon that changes occasionally need to be applied to the schema as the application requirements change. For example, we may decide that another data item needs to be stored for each record in a file, such as adding the Date_of_birth to the STUDENT schema in Figure 2.1 (in slide number 24). This is known as **schema evolution**. Most modern DBMSs include some operations for schema evolution that can be applied while the database is operational.

Schemas and Instances

- **Logical Schema** – the overall logical structure of the database
  - Example: The database consists of information about a set of customers and accounts in a bank and the relationship between them
    - Analogous to type information of a variable in a program
- **Physical schema**– the overall physical structure of the database
- **Instance** – the actual content of the database at a particular point in time
  - Analogous to the value of a variable

Three-Schema Architecture and Data Independence

- It divides the system into three levels of abstraction: the *internal* or *physical level*, the *conceptual level*, and the *external* or *view level*.
- The goal of the three schema architecture is to separate the user applications and the physical database.

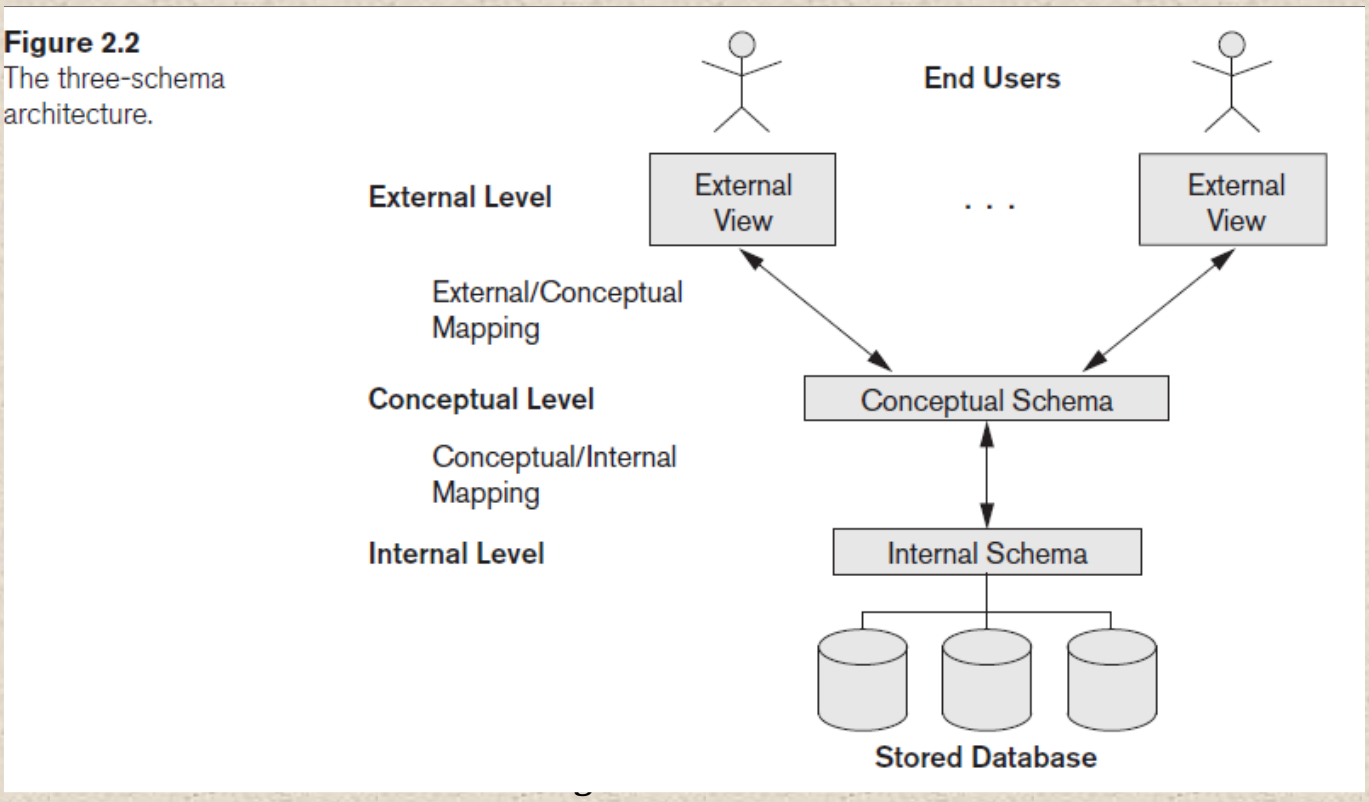


Figure 2.2
The three-schema architecture.

Three-Schema Architecture and Data Independence

- 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.
- 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 conceptual data model.

Three-Schema Architecture and Data Independence

- 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**.

Levels of Abstraction

- **Physical level:** describes how a record (e.g., instructor) is stored. It describes the physical storage structure of the data in the database.
- **Logical level:** describes data stored in database, and the relationships among the data. It describes the structure of whole database and hides details of physical storage structure. It concentrates on describing entities, data types, relationships, attributes and constraints.

```
type instructor = record
        ID : string;
        name : string;
        dept_name : string;
        salary : integer;
            end;
```

- **View level:** application programs hide details of data types. It includes a number of user views and hence is guided by the end user requirement. It describes only those part of the database in which the users are interested and hides rest of all from those users. Views can also hide information (such as an employee's salary) for security purposes.

Data Independence

The three schema architecture further explains the concept of **data independence**, the capacity to change the schema at one level without having to change the schema at the next higher level.

- **Logical Data Independence**
- **Physical Data Independence**

- **Logical Data Independence:** The capacity to change the conceptual schema without having to change the external schemas and their associated 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).

Physical Data Independence: The capacity to change the internal schema without having to change the conceptual schema. Applications depends on logical schema. In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

For example, the internal schema may be changed when certain file structures are reorganized or new indexes are created to improve database performance

Data Independence…….

- When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence. The higher-level schemas themselves are unchanged.

- Hence, the application programs need not be changed since they refer to the external schemas.

Database Languages

- **Database Language: L**anguage used to interrogate and process data in a relational database
  - **Data definition language (DDL):** Set of statements that describe a database structure (all record types and data set types). DDL statements include creating databases, tables etc. The DDL is used to specify the conceptual schema only.
  - **DCL (Data Control Language):** DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.
  - **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.* In relational DBMSs, SQL is used in the role of VDL to define user or application **views** as results of predefined queries.

Database Languages

- **Data Manipulation Language (DML):** Instructions used with higher-level programming languages to query the contents of the database, store or update information, and develop database applications. DML statements include accessing data from database.
- There are two main types of DMLs.
  - **High-level of non procedural**
  - **Low-level or procedural**
- A **high-level** or **nonprocedural** DML can be used on its own to specify complex database operations concisely. 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. In the latter case, DML statements must be identified within the program so that they can be extracted by a precompiler and processed by the DBMS.

Database Languages

- 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. Therefore, it needs to use programming language constructs, such as looping, to retrieve and process each record from a set of records. This type of DML typically retrieves individual records from the database and processes each separately. In this language, the looping, branching etc. Statements are used 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. The programmers use the low-level DML.
- **High-level DMLs, such as SQL**, can specify and retrieve many records in a single DML statement; therefore, they are 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; therefore, such languages are also called **declarative**.

Database Languages

- 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**.
- On the other hand, a high-level DML used in a standalone interactive manner is called a **query language**

The Database System Environment

- A database system is partitioned into modules that deal with each of the responsibilities of the overall system.

Database System Environment

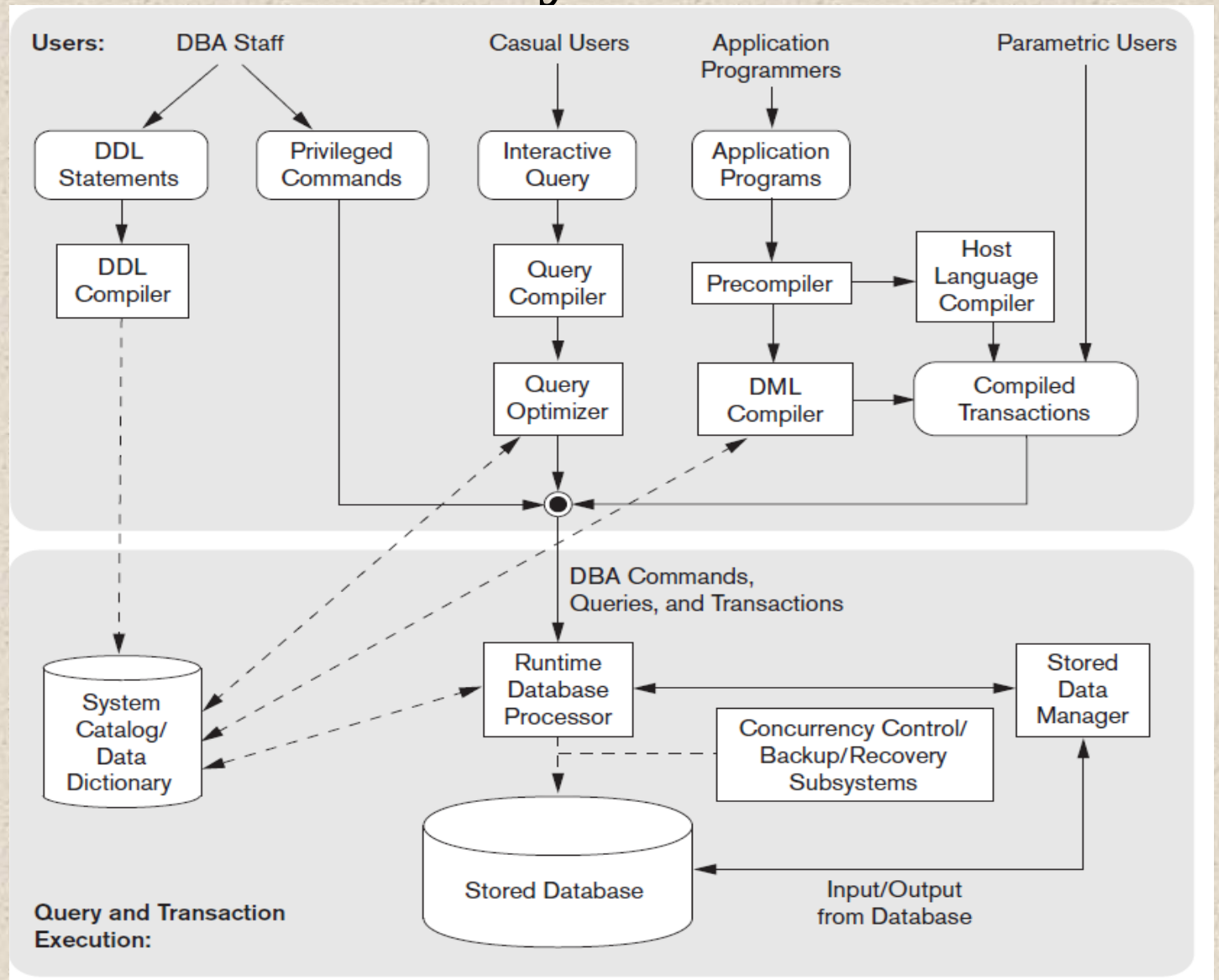


Figure 2.3
Component modules of a DBMS and their interactions.

The Database System Environment

- **DBMS 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 read/write. Many DBMSs have their own **buffer management** module to schedule disk read/write, because management of buffer storage has a considerable effect on performance. Reducing disk read/write improves performance considerably. 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 **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 and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints.
- Casual users and persons with occasional need for information from the database interact using the **interactive query** interface. These queries are parsed and validated for correctness of the query syntax, the names of files and data elements, and so on by a **query compiler** that compiles into an internal form.

The Database System Environment

- **DBMS Component Modules:**
- This internal query is subjected to query optimization. Among other things, the **query optimizer** is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of efficient search algorithms during execution.
- Application programmers write programs in host languages such as Java, C, or C++ that are submitted to a precompiler. 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 **runtime database processor** executes (1) the privileged commands, (2) the executable query plans, and (3) the canned transactions with runtime parameters. It works with the **system catalog** and may update it with statistics. It also works with the **stored data manager**, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory. **Concurrency control** and **backup and recovery systems** are integrated into the working of the runtime database processor for purposes of transaction management.

The Database System Environment

- **Database System Utilities:** Most DBMSs have **database utilities** that help the DBA manage the database system.
  - **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.
  - **Backup.** A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape or other mass storage medium. The backup copy can be used to restore the database in case of catastrophic disk failure.
  - **Database storage reorganization.** This utility can be used to reorganize a set of database files into different file organizations and create new access paths 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 or whether to add or drop indexes to improve performance.

The Database System Environment

- **Tools, Application Environments, and Communications Facilities:** Other tools are often available to database designers, users, and the DBMS.
  - 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**.
- **Application development environments**, such as PowerBuilder (Sybase) or JBuilder (Borland), have been 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, GUI 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 personal computers

Centralized and Client/Server Architectures for DBMSs

- Database systems structure:
  - Centralized databases
    - One to a few cores, shared memory
  - Client-server,
    - One server machine executes work on behalf of multiple client machines.

Centralized and Client/Server Architectures for DBMSs

- **Centralized Database:** The DBMS itself was still a **centralized** DBMS in which all the DBMS functionality, application program execution, and user interface processing were carried out on one machine.

Centralized and Client/Server Architectures for DBMSs

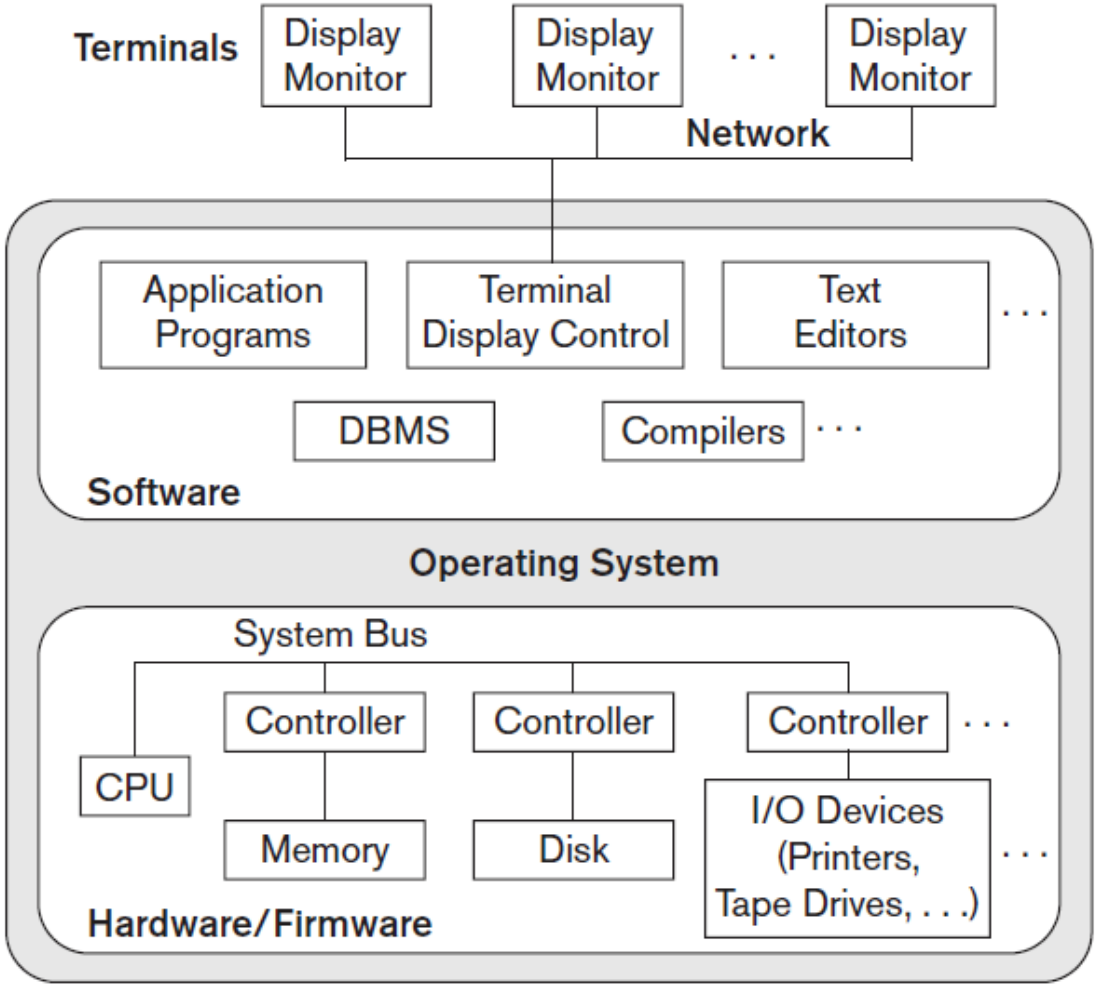


Figure 2.4
A physical centralized architecture.

Centralized and Client/Server Architectures for DBMSs

- **Basic Client Server Architecture:**
- The **client/server architecture** was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, database servers, Web servers, e-mail servers, and other software and equipment are connected via a network.
- The idea is to define **specialized servers** with specific functionalities. For example, it is possible to connect a number of PCs or small workstations as clients to a **file server** that maintains the files of the client machines. Another machine can be designated as a **printer server** by being connected to various printers; all print requests by the clients are forwarded to this machine.
- **Web servers** or **e-mail servers** also fall into the specialized server category.

Centralized and Client/Server Architectures for DBMSs

- **Basic Client Server Architecture:**
- The resources provided by specialized servers can be accessed by many client machines. The **client machines** provide the user with the appropriate interfaces to utilize these servers, as well as with local processing power to run local applications.
- A **client** in this framework is typically a user machine that provides user interface capabilities and local processing. When a client requires access to additional functionality—such as database access—that does not exist at the client, it connects to a server that provides the needed functionality. A **server** is a system containing both hardware and software that can provide services to the client machines, such as file access, printing, archiving, or database access.

Centralized and Client/Server Architectures for DBMSs

- **Basic Client Server Architecture:**

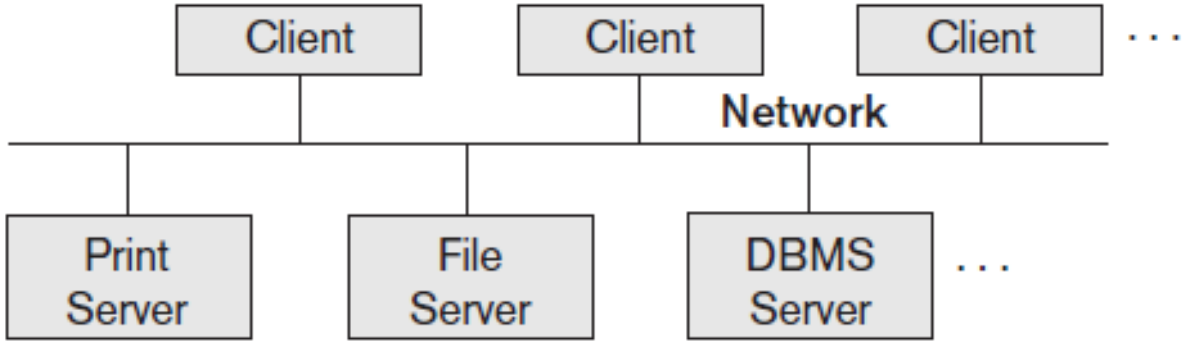


Figure 2.5
Logical two-tier client/server architecture.

Centralized and Client/Server Architectures for DBMSs

- Underlying client/server framework database applications are usually partitioned into two, three or n-parts.
  - **Two-tier architecture --** the application resides at the client machine, where it invokes database system functionality at the server machine
  - The user interface programs and application programs can run on the client side. When DBMS access is required, the program establishes a connection to the DBMS (which is on the server side); once the connection is created, the client program can communicate with the DBMS.

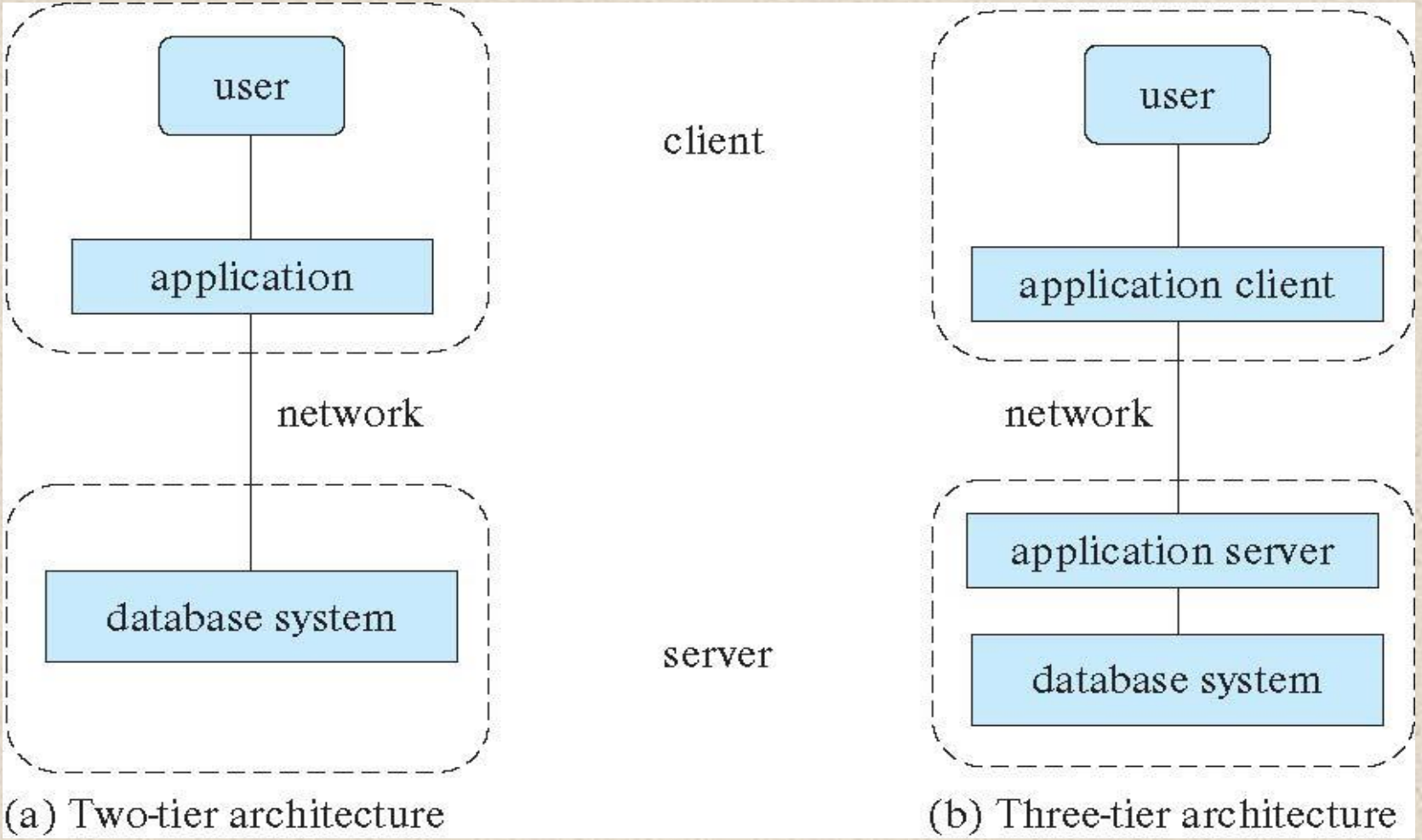
Centralized and Client/Server Architectures for DBMSs

- **Three-tier architecture --** the **client machine acts as a front end** and does not contain any direct database calls. The client end communicates with an **application server**, usually through a forms interface. The application server in turn communicates with **a database system** to access data.
- This intermediate layer or **middle tier** is called the **application server** or the **Web server**, depending on the application. This server plays an intermediary role by running application programs and storing business rules (procedures or constraints) that are used to access data from the database server. It can also improve database security by checking a client's credentials before forwarding a request to the database server. Clients contain **user interfaces and web browsers**. The **intermediate server** accepts requests from the client, processes the request and sends database queries and commands to the **database server**, and then acts as a conduit for passing (partially) processed data from the database server to the clients, where it may be processed further and filtered to be presented to the users. Thus, the *user interface*, *application rules*, and *data access* act as the three tiers.

Client/Server Database Architecture

– **n-tier architecture --** the client machine acts as a front end and does not contain any direct database calls.
  - The client end communicates with an application server, usually through a forms interface.
  - The application server in turn communicates with n-tiers/layers of servers

Two-tier and three-tier architectures



Relational Algebra

- **The relational algebra** is a procedural query language. A relational algebra query describes a procedure for computing the output relation from the input relations by applying the relational algebra operators.
- The relational algebra defines a set of operators from set theory as *Union*, *Intersection*, *Set Difference*, *Cartesian Product*. Similarly, other operations as *Select*, *Project & Join* are incorporated with in the algebra operations. These operations can be categorized as unary or binary according as the number of relation on which they operate on.
- **Importance of Relational Algebra:**

  The relational algebra is very important for several reasons:
  - ***First***, it provides a formal foundation for relational model operations.
  - ***Second*** and perhaps more important, it is used as a basis for implementing and optimizing queries in relational DBMS (RDBMS).
  - ***Third***, some of its concepts are incorporated into the SQL standard query language for RDBMS

Codd's Rule

- Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.
- Rule 1: Information Rule
  - The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.
- Rule 2: Guaranteed Access Rule
  - Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.
- Rule 3: Systematic Treatment of NULL Values
  - The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following − data is missing, data is not known, or data is not applicable.

Codd's Rule

- Rule 4: Active Online Catalog
  - The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.
- Rule 5: Comprehensive Data Sub-Language Rule
  - A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.
- Rule 6: View Updating Rule
  - All the views of a database, which can theoretically be updated, must also be updatable by the system

Codd's Rule

- Rule 7: High-Level Insert, Update, and Delete Rule
  - A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.
- Rule 8: Physical Data Independence
  - The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.
- Rule 9: Logical Data Independence
  - The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application.

Codd's Rule

- Rule 10: Integrity Independence
  - A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.
- Rule 11: Distribution Independence
  - The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.
- Rule 12: Non-Subversion Rule
  - If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.