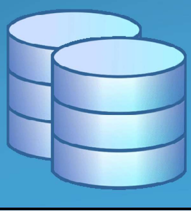
CSIE30600/CSIEB0290
Database Systems

Lecture 2: Models and Architecture



Outline

- Data Abstraction
- Data Models and Categories
- Schemas, Instances, and States
- Three-level (Three-Schema) Architecture
- Data Independence
- DB Languages and Interfaces
- DB System Utilities and Tools
- DB System Environment
- Centralized and Client-Server Architectures
- Classification of DBMSs

Data Abstraction

- Suppression of details of data organization and storage
- Highlighting of the essential features for an improved understanding of data
- Key to the success of database systems
- Useful for other domains as well

Levels of Abstraction on Data

- **Physical** level**:** describes how a record is stored.
- **Logical** level**:** describes data stored in database, and the relationships among the data.

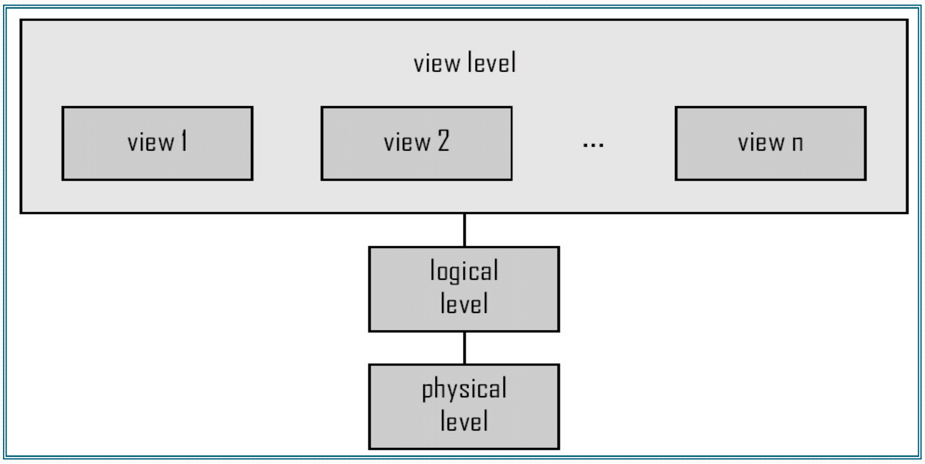
  type *customer* = **record**

  customer_id : string;
  customer_name : string;
  customer_street : string;
  customer_city : integer;

  end;
- **View** level**:** application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.

Three-Level Architecture

An architecture for a database system



Data Models

- **Data model**: for data abstraction
  - A set of concepts to describe the structure of a DB, the operations on these structures, and certain constraints that the DB should obey.
- Data model **structure**:
  - Constructs are used to define the DB structure
    - Elements (and ***data types***)
    - Groups of elements (e.g. ***entity, record, table***)
    - Relationships among such groups
- Data model **constraints**:
  - specify some restrictions on **valid** data
  - must be enforced at ALL times

Data Models (cont.)

- Data model **operations**:
  - used for specifying database retrievals and updates by referring to the constructs of the data model
- Operations on the data model may include:
  - basic model operations (e.g. generic insert, delete, update)
  - user-defined operations (e.g. compute_student_gpa, update_inventory)

Categories of Data Models

- **Conceptual** (high-level, semantic) data models**:**
  - Close to the way many users perceive data. (Also called ***entity-based*** or ***object-based*** data models.)
- **Physical** (low-level, internal) data models**:**
  - Describe details of how data is stored in the computer.
- **Representational** (record-oriented, implementation) data models**:**
  - Fall between the above two, used by many commercial DBMS implementations (e.g. relational data models used in many commercial systems).
- **Self-Describing** data models:
  - Combine the description of data with the data values. (e.g. XML, key-value stores, some NoSQL systems)

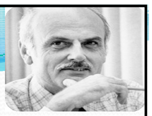
Modeling Elements

- **Entity**
  - Represents a real-world object or concept
- **Attribute**
  - Represents some property of interest
  - Further describes an entity
- **Relationship** among two or more entities
  - Represents an association among the entities
  - Represents constraints on the relationships

Data Models: Examples

- Relational model
- Entity-Relationship data model (mainly for database design)
- Physical data model (for data storage)
- Object-based data models (Object-oriented and Object-relational)
- Semistructured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model

Relational Model



Ted Codd
Turing Award 1981

- Relational model represents data in tabular form

Attributes(Columns)

Table

Rows

customer_id	customer_name	customer_street	customer_city	account_number
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-101
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-201
677-89-9011	Hayes	3 Main St.	Harrison	A-102
182-73-6091	Turner	123 Putnam St.	Stamford	A-305
321-12-3123	Jones	100 Main St.	Harrison	A-217
336-66-9999	Lindsay	175 Park Ave.	Pittsfield	A-222
019-28-3746	Smith	72 North St.	Rye	A-201

A Sample RDB

customer-id	customer-name	customer-street	customer-city
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
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

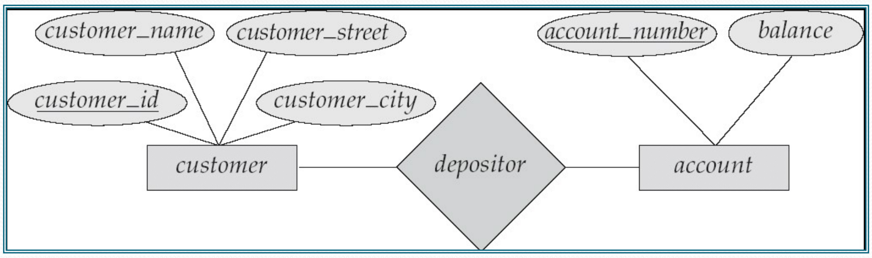
(b) The *account* table

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

The Entity-Relationship Model

- Models an enterprise as a collection of entities and relationships
  - **Entity**: a "thing" or "object" in the enterprise that is distinguishable from other objects, described by a set of ***attributes***
  - **Relationship**: an association among several entities
- Represented diagrammatically by an entity-relationship diagram:



Physical Data Models

- Describe how data is stored as files in the computer
- Access path
  - Structure that makes the search for particular database records efficient
- Index
  - Example of an access path
  - Allows direct access to data using an index term or a keyword

Object-Relational Data Models

- Extend the relational data model by including object orientation and constructs to deal with added data types.
- Allow attributes of tuples to have complex types, including non-atomic values such as nested relations.
- Preserve relational foundations, in particular the declarative access to data, while extending modeling power.
- Provide upward compatibility with existing relational languages.

XML: Extensible Markup Language

- Defined by the **WWW Consortium** (**W3C**)
- Originally intended as a document markup language not a database language
- The ability to specify new tags, and to create nested tag structures made XML a great way to exchange data, not just documents
- XML has become the basis for all new generation **data interchange formats**.
- A wide variety of tools is available for parsing, browsing and querying XML documents/data

Schemas and Instances

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

Schemas

- Database Schema:
  - The description of a database.
  - Includes descriptions of the database structure, data types, and the constraints on the database.
- Schema Diagram:
  - An illustrative display of (most aspects of) a database schema.
- Schema Construct:
  - A component of the schema or an object within the schema, e.g., STUDENT, COURSE.

Database State (Instance)

- Database State:
  - The content (actual data) stored in a database at a ***particular moment in time***.
  - This includes the collection of all the data in the database.
  - Also called database instance (or occurrence or snapshot).
  - The term *instance* is also applied to individual database components, e.g. *record instance, table instance, entity instance*

Database State (cont.)

- Initial Database State:
  - Refers to the database state when it is initially loaded into the system.
- Valid State:
  - A state that satisfies the structure and constraints of the database.

Schema vs. State

- Distinction
  - The ***database schema*** changes very infrequently.
  - The ***database state*** changes every time the database is updated.
- **Schema** is also called intension.
- **State** is also called extension.
- Schema evolution
  - Changes applied to schema as application requirements change

Example: Database Schema

Figure 2.1
Schema diagram for the database in Figure 1.2.

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Example: Database State

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	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	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.

Three-Schema Architecture

- Proposed to support DBMS characteristics of:
  - **Program data independence**.
  - Support of **multiple views** of data.
- Defines DBMS schemas at ***three*** levels:
  - **Internal schema** at the **internal level** to describe data **storage structures** and **access paths**. Typically uses a ***physical*** data model.
  - **Conceptual schema** at the **conceptual level** to describe the structure and constraints for the *whole* database. Uses a ***conceptual*** or an ***implementation*** data model.
  - **External schemas** at the **external level** to describe the various **user views**. Usually uses the same data model as the conceptual level.

Three-Schema Architecture (cont.)

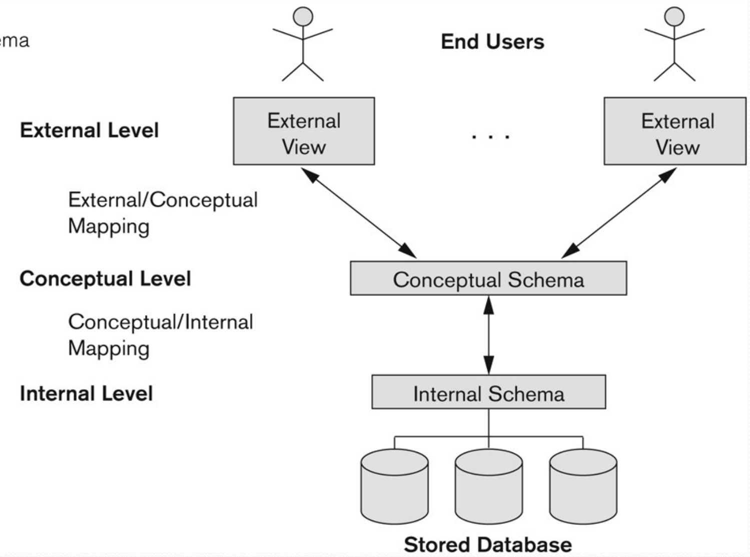
- **Many** views, **single** conceptual (logical) schema and physical schema.
- **Views** describe how users see the data.
- **Conceptual schema** defines logical structure
- **Physical schema** describes the files and indexes used to store the data.

Schema Mapping

- **Mappings** among schema levels are also needed. Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.
- Three-level architecture is not explicitly used in commercial DBMS products, but has been useful in explaining database system organization.

Three-Schema Architecture

Figure 2.2
The three-schema architecture.



Data Independence

- Capacity to change the schema at one level of a database system without having to change the schema at the next higher level
- **Logical Data Independence**: can change the conceptual schema without changing the external schemas and their application programs.
- **Physical Data Independence**: can change the internal schema without changing the conceptual schema.
- Applications depend on the 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.

Data Independence (cont.)

- When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed.
- The higher-level schemas themselves are unchanged. Hence, the application programs need not be changed since they refer to the external schemas.

DBMS Languages

- **Data Definition Language (DDL)**: Used by the DBA and database designers to specify the conceptual schema. In many DBMSs, the DDL is also used to define internal and external schemas (views).
- In some DBMSs, separate storage definition language (SDL) and view definition language (VDL) are used to define internal and external schemas.

DBMS Languages (cont.)

- **Data Manipulation Language (DML):** Used to specify database retrievals and updates.
- DML commands (data sub-language) can be **embedded** in a general-purpose programming language (host language), such as COBOL, PL/1 or PASCAL.
- Alternatively, stand-alone DML commands can be applied directly (**query language**).

Data Definition Language (DDL)

- Specification notation for defining the database schema

  Example: **create table** *account* (
  account-number **char**(10),
  balance **integer**)
- **DDL compiler** generates a set of tables stored in a **data dictionary**

Data Dictionary

- Data dictionary contains **metadata** (i.e., data about data)
  - Database schema
  - Data *storage and definition* language
    - Specifies the storage structure and access methods used
  - Integrity constraints
    - Domain constraints
    - Referential integrity (**references** constraint in SQL)
    - Assertions
  - Authorization

Data Manipulation Language (DML)

- For **accessing** and **manipulating** the data organized by the appropriate data model
  - DML also known as **query language**
- Two classes of languages
  - **Procedural** – user specifies what data is required and how to get those data
  - **Declarative** (nonprocedural) – user specifies what data is required without specifying how to get those data
- SQL is the most widely used query language

SQL

- **SQL(Structured Query Language):** widely used non-procedural language
  - Example: Find the name of the customer with id 192-83-7465

    select *customer.customer_name*
    from *customer*
    where *customer.customer_id* = '192-83-7465'
  - Example: Find the balances of all accounts held by the customer with id 192-83-7465

    select *account.balance*
    from *depositor, account*
    where *depositor.customer_id* = '192-83-7465' **and**
    depositor.account_number =
    account.account_number

SQL in Application Programs

- Application programs generally access databases through:
  - Language extensions to allow **embedded SQL**
  - **Application program interface(API)** (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database

DBMS Interfaces

- Stand-alone query language interfaces
  - Example: Entering SQL queries at the DBMS interactive SQL interface (e.g. SQL*Plus in ORACLE)
- Programmer interfaces for embedding DML in programming languages
- User-friendly interfaces
  - Menu-based, forms-based, graphics-based, etc.
- Mobile interfaces
  - Allowing users to perform transactions using mobile apps

DBMS Programming Language Interfaces

- Programmer interfaces for embedding DML in a programming languages:
  - Embedded Approach: e.g. embedded SQL (for C, C++, etc.), SQLJ (for Java)
  - Procedure Call Approach: e.g. JDBC for Java, ODBC for other programming languages
  - Database Programming Language Approach: e.g. ORACLE has PL/SQL, a programming language based on SQL; language incorporates SQL and its data types as integral components
  - Scripting Languages: e.g. JavaScript(client-side scripting) and PHP(server-side scripting) are used to write database programs

User-Friendly DBMS Interfaces

- Menu-based, popular for browsing on the web
- Forms-based, designed for naïve users
- Graphics-based
  - (Point and Click, Drag and Drop, etc.)
- Natural language: requests in written English
- Combinations of the above:
  - For example, both menus and forms used extensively in Web database interfaces

Other DBMS Interfaces

- Natural language: free text as a query
- Speech as Input and Output
- Web Browser as an interface
- Parametric interfaces, e.g., bank tellers using function keys.
- Interfaces for the DBA:
  - Creating user accounts, granting authorizations
  - Setting system parameters
  - Changing schemas or access paths

Database System Utilities

- To perform certain functions such as:
  - Loading data stored in files into a database. Includes data conversion tools.
  - Backup the database periodically.
  - Reorganizing database file structures.
  - Report generation utilities.
  - Performance monitoring utilities.
  - Other functions, such as sorting, user monitoring, data compression, etc.

Other Tools

- Data dictionary/repository:
  - Used to store schema descriptions and other information such as design decisions, application program descriptions, user information, usage standards, etc.
  - Active data dictionary is accessed by DBMS software and users/DBA.
  - Passive data dictionary is accessed by users/DBA only.

Other Tools

- Application development and CASE (computer-aided software engineering) tools
  - **Examples**: GitHub, Google Cloud Platform(Google), AWS Cloud9(Amazon), Azure(Microsoft), IntelliJ IDEA(JetBrains), JDeveloper(Oracle), Apache NetBeans(Apache), Anaconda Distribution(Anaconda), …
- Communication software

Database Engine

- A database system is partitioned into modules that deal with each of the responsibilities of the overall system.
- The functional components of a database system can be divided into
  - The storage manager,
  - The query processor component,
  - The transaction management component.

Storage Management

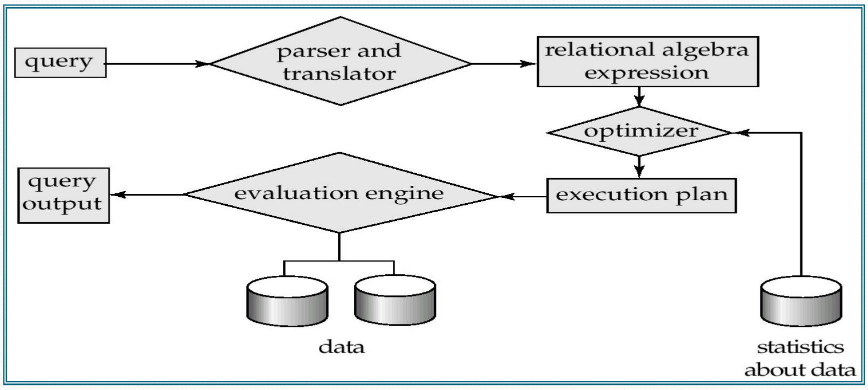
- **Storage manager** is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible for:
  - Interaction with the OS file manager
  - Efficient data storing, retrieving and updating
- Issues:
  - Access, authorization, integrity manager
  - File manager, buffer manager
  - Data dictionary (stores metadata)
  - Indexing and hashing

Query Processor

- The query processor components include:
  - DDL interpreter -- interprets DDL statements and records the definitions in the data dictionary.
  - DML compiler -- translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
    - The DML compiler performs query optimization; that is, it picks the lowest cost evaluation plan from among the various alternatives.
  - Query evaluation engine -- executes low-level instructions generated by the DML compiler.

Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Query Processing (Cont.)

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation
- Cost difference between a good and a bad way of evaluating a query can be enormous!
- Need to estimate the cost of operations
  - Depends critically on statistical information about relations which the database must maintain
  - Need to estimate statistics for intermediate results to compute cost of complex expressions

Transaction Management

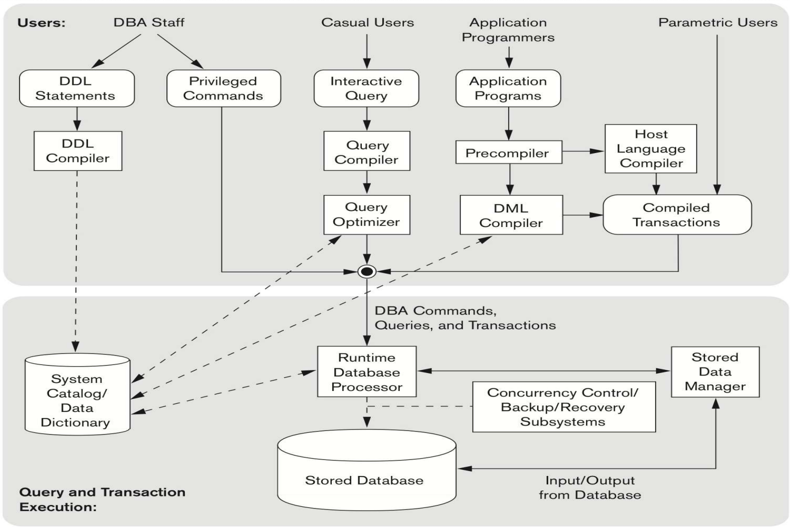
- A transaction is a collection of operations that performs a single logical function in a database application
- Transaction-management component ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- Concurrency-control manager controls the interaction among the concurrent transactions, to ensure the consistency of the database.

The Database System Environment

- DBMS component modules
  - Buffer management
  - Stored data manager
  - DDL compiler
  - Interactive query interface (Query compiler, optimizer)
  - Precompiler
  - Runtime database processor
  - System catalog
  - Concurrency control system
  - Backup and recovery system

Typical DBMS Component Modules

Component modules of a DBMS and their interactions.



Database Design

- The process of designing the structure of a DB
- **Logical Design** – Deciding on the database schema. Database design requires that we find a "good" collection of relation schemas.
  - Business decision – What information should we record in the database?
  - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- **Physical Design** – Deciding on the physical layout of the database

Database Architecture

- The architecture of a database systems is greatly influenced by the underlying computer system on which the database is running
- Centralized
- Client-server
- Parallel (multi-processor)
- Distributed

Centralized DBMS Architectures

- Combines everything into single system including- DBMS software, hardware, application programs, and user interface processing software.
- User can still connect through a remote terminal – however, all processing is done at centralized site.

A Physical Centralized Architecture

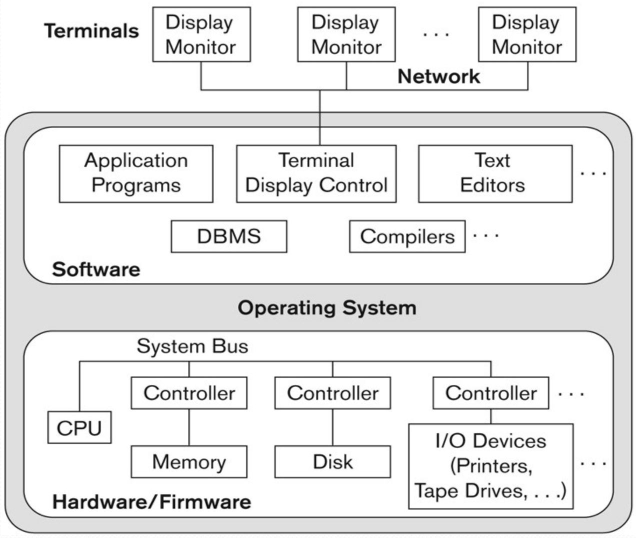


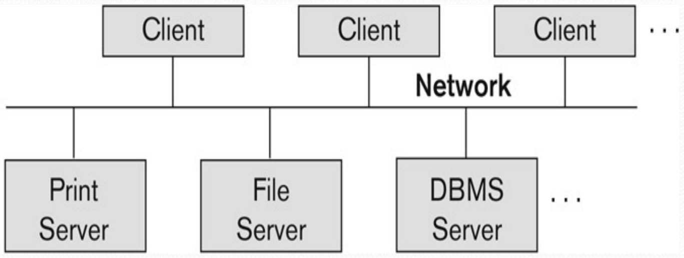
Figure 2.4
A physical centralized architecture.

Basic 2-tier Client-Server Architectures

- Specialized servers with specialized functions
  - Print server
  - File server
  - DBMS server
  - Web server
  - Email server
- Clients can access the specialized servers as needed

Logical 2-tier Client-Server Architecture

Figure 2.5
Logical two-tier client/server architecture.



Physical 2-tier Architecture

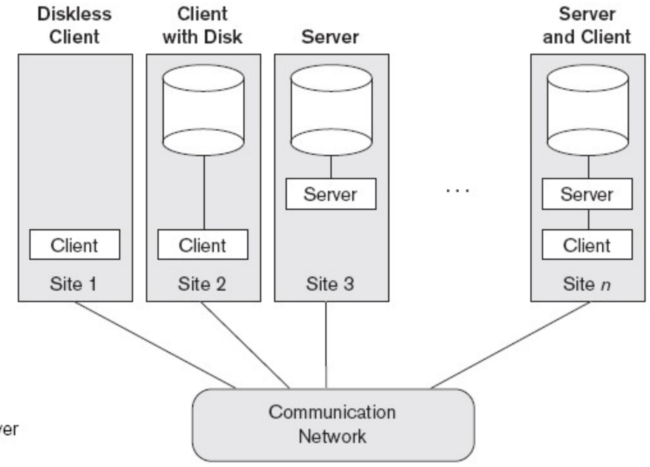


Figure 2.6
Physical two-tier client/server architecture.

Clients

- Provide appropriate interfaces through a client software module to access and utilize the various server resources.
- Clients may be diskless machines or PCs or Workstations with disks with only the client software installed.
- Connected to the servers via some form of a network.
  - (LAN: local area network, wireless network, etc.)

DBMS Server

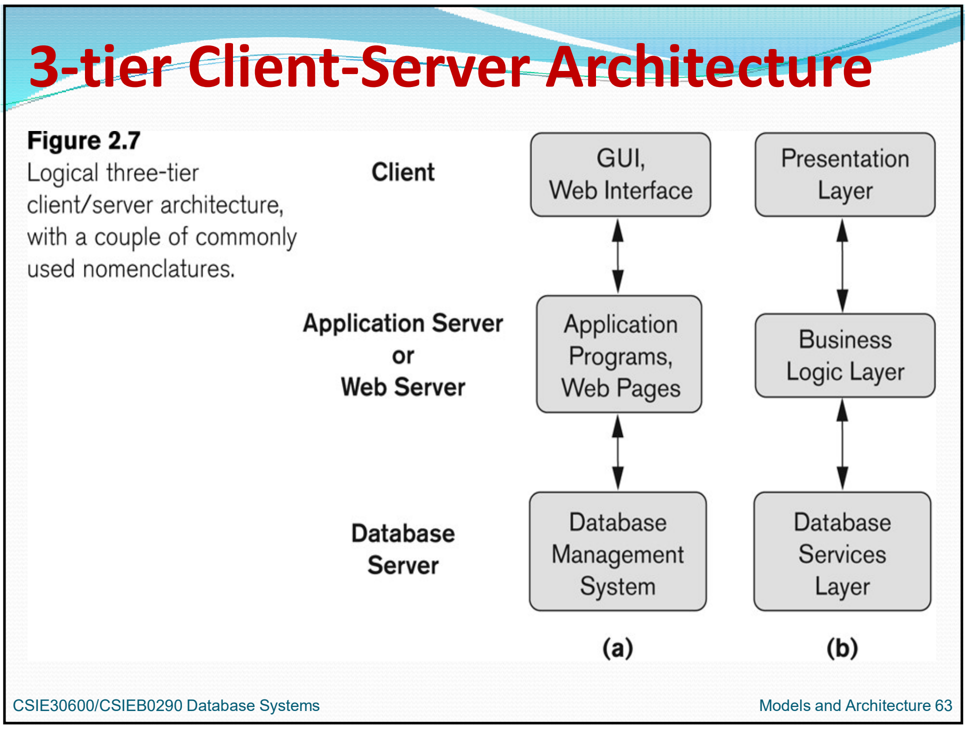
- Provides database query and transaction services to the clients
- Relational DBMS servers are often called SQL servers, query servers, or transaction servers
- Applications running on clients utilize an Application Program Interface (API) to access server databases via standard interface such as ODBC(Open Database Connectivity) and JDBC.
- Client and server must install appropriate client and server module software for ODBC or JDBC
- More about this in later lectures.

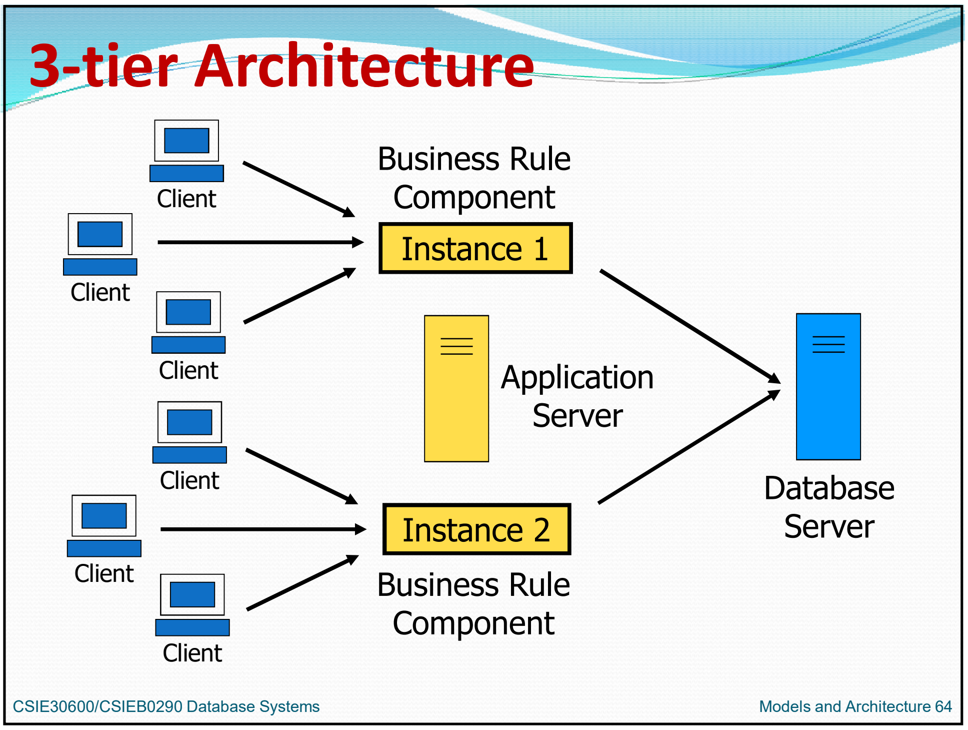
Characteristics of 2-tier Client-Server Architecture

- A client program may connect to several DBMSs, sometimes called the data sources.
- In general, data sources can be files or other non-DBMS software that manages data.
- Other variations of clients are possible: e.g., in some object DBMSs, more functionality is transferred to clients including data dictionary functions, optimization and recovery across multiple servers, etc.

3-Tier Client-Server Architecture

- Common for Web applications
- Intermediate layer called application server or Web server:
  - Stores the Web connectivity software and the business logic part of the application used to access the corresponding data from the database server
  - Acts like a conduit(管道) for sending partially processed data between the database server and the client.
- 3-tier architecture can enhance security:
  - Database server only accessible via middle tier
  - Clients cannot directly access database server
  - Clients contain user interfaces and Web browsers
  - Client is typically a PC or a mobile device


3-tier Client-Server Architecture
Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.
Client
GUI, Web Interface
Presentation Layer
Application Server or Web Server
Application Programs, Web Pages
Business Logic Layer
Database Server
Database Management System
Database Services Layer
(a)
(b)
CSIE30600/CSIEB0290 Database Systems
Models and Architecture 63


3-tier Architecture
Client
Client
Client
Client
Client
Client
Business Rule Component
Instance 1
Application Server
Database Server
Instance 2
Business Rule Component
CSIE30600/CSIEB0290 Database Systems
Models and Architecture 64

Characteristics 3-Tier Architecture

- **Advantages**:
  - Moving business rule components to an application server can boost performance
  - Load balancing and fault tolerance with multiple application servers
  - Changes to business rules only affect a small number of application servers
  - Better code encapsulation
- **Problem**: can generate a lot of network activity (why?)

Multi-Tier (n-Tier) Architecture

- User Interface Services Tier
  - handles UI logic
- UI-Oriented Business Rule Services Tier
  - handles user interface related business rule logic
  - validation of input
- Data-Oriented Business Rule Services Tier
  - data manipulation and integration
  - can integrate SQL database
- Data Persistence Services Tier
  - handles storage and retrieval of data

Characteristics of n-Tier Model

- The key idea is to keep the services physically close to the data they work with.
- UI-oriented business rule components can be placed on the client.
- Data-oriented business rule components are deployed on database or application server.
- Scale well
- Flexible about placement and presence of application servers.

Classification of DBMSs

- Based on the data model used
  - Legacy: Network, Hierarchical
  - Currently Used: Relational, Object-oriented, Object-relational
  - Recent Technologies: XML, Key-value store, NoSQL, document based, column-based, graph-based …
- Other classifications
  - Single-user (typically used with personal computers) vs. multi-user (most DBMSs).
  - Centralized (uses a single computer with one database) vs. distributed (uses multiple computers, multiple databases)
  - Open source vs. commercial

Variations of Distributed DBMSs (DDBMSs)

- Homogeneous DDBMS
- Heterogeneous DDBMS (Federated or Multidatabase Systems)
- Distributed Database Systems have now come to be known as client-server based database systems because:
  - They do not support a totally distributed environment, but rather a set of database servers supporting a set of clients.

Cost Considerations for DBMSs

- Cost Range: from free open-source systems to configurations costing millions of dollars
- Examples of free relational DBMSs: MySQL, PostgreSQL, others
- Commercial DBMS offer additional specialized modules, e.g. time-series module, spatial data module, document module, XML module
  - These offer additional specialized functionality when purchased separately
  - Sometimes called cartridges (e.g., in Oracle) or blades
- Different licensing options: site license, maximum number of concurrent users (seat license), single user, etc.

Other Considerations

- Type of access paths within database system
  - E.g.- inverted indexing based (ADABAS is one such system).Fully indexed databases provide access by any keyword (used in search engines)
- General Purpose vs. Special Purpose
  - E.g.- Airline Reservation systems or many others- reservation systems for hotel/car etc. are special purpose OLTP (Online Transaction Processing Systems)

Summary

- Data Abstraction and Three-level Architecture
- Data Models and Their Categories
- Schemas, Instances, and States
- Three-Schema Architecture
- Data Independence
- DBMS Languages and Interfaces
- Database System Utilities and Tools
- Database System Environment
- Centralized and Client-Server Architectures
- Classification of DBMSs

Assignment 1

- Textbook exercises: 1.2, 1.6, 1.10, 1.12, 1.15
- Due date: **Oct 14, 2020**