Chapter 2

Database System Concepts and Architecture
Outline

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

- **Data Model:**
  - A set of concepts to describe the *structure* of a database, the *operations* for manipulating these structures, and certain *constraints* that the database should obey.

- **Data Model Structure and Constraints:**
  - Constructs are used to define the database structure
  - Constructs typically include *elements* (and their *data types*) as well as groups of elements (e.g. *entity*, *record*, *table*), and *relationships* among such groups
  - Constraints specify some restrictions on valid data; these constraints must be enforced at all times
Data Models (continued)

- **Data Model Operations:**
  - These operations are 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) and *user-defined operations* (e.g., compute_student_gpa, update_inventory)
Categories of Data Models

- **Conceptual (high-level, semantic) data models:**
  - Provide concepts that are close to the way many users perceive data.
    - (Also called *entity-based* or *object-based* data models.)

- **Physical (low-level, internal) data models:**
  - Provide concepts that describe details of how data is stored in the computer. These are usually specified in an ad-hoc manner through DBMS design and administration manuals.

- **Implementation (representational) data models:**
  - Provide concepts that fall between the above two, used by many commercial DBMS implementations (e.g. relational data models used in many commercial systems).
Schemas versus Instances

- **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.
Schemas versus Instances

- **Database State:**
  - The 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 Schema vs. Database State

- **Database State:**
  - Refers to the *content* of a database at a moment in time.

- **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.
Database Schema vs. Database State (continued)

- 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**.
**Example of a Database Schema**

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>COURSE</th>
<th>PREREQUISITE</th>
<th>SECTION</th>
<th>GRADE_REPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Student_number</td>
<td>Class</td>
<td>Major</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COURSE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Course_name</td>
<td>Course_number</td>
<td>Credit_hours</td>
<td>Department</td>
<td></td>
</tr>
<tr>
<td>PREREQUISITE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Course_number</td>
<td>Prerequisite_number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECTION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Section_identifier</td>
<td>Course_number</td>
<td>Semester</td>
<td>Year</td>
<td>Instructor</td>
</tr>
<tr>
<td>GRADE_REPORT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Student_number</td>
<td>Section_identifier</td>
<td>Grade</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.1**  
Schema diagram for the database in Figure 1.2.
Example of a database state

<table>
<thead>
<tr>
<th>COURSE</th>
<th>Course_name</th>
<th>Course_number</th>
<th>Credit_hours</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro to Computer Science</td>
<td>CS1310</td>
<td>4</td>
<td>CS</td>
<td></td>
</tr>
<tr>
<td>Data Structures</td>
<td>CS3320</td>
<td>4</td>
<td>CS</td>
<td></td>
</tr>
<tr>
<td>Discrete Mathematics</td>
<td>MATH2410</td>
<td>3</td>
<td>MATH</td>
<td></td>
</tr>
<tr>
<td>Database</td>
<td>CS3380</td>
<td>3</td>
<td>CS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SECTION</th>
<th>Section_identifier</th>
<th>Course_number</th>
<th>Semester</th>
<th>Year</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
<td>MATH2410</td>
<td>Fall</td>
<td>04</td>
<td>King</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>CS1310</td>
<td>Fall</td>
<td>04</td>
<td>Anderson</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>CS3320</td>
<td>Spring</td>
<td>05</td>
<td>Knuth</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>MATH2410</td>
<td>Fall</td>
<td>05</td>
<td>Chang</td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>CS1310</td>
<td>Fall</td>
<td>05</td>
<td>Anderson</td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>CS3380</td>
<td>Fall</td>
<td>05</td>
<td>Stone</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GRADE REPORT</th>
<th>Student_number</th>
<th>Section_identifier</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>112</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>119</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>85</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>92</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>102</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>135</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 1.2**  
A database that stores student and course information.

<table>
<thead>
<tr>
<th>PREREQUISITE</th>
<th>Course_number</th>
<th>Prerequisite_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS3380</td>
<td>CS3320</td>
<td></td>
</tr>
<tr>
<td>CS3380</td>
<td>MATH2410</td>
<td></td>
</tr>
<tr>
<td>CS3320</td>
<td>CS1310</td>
<td></td>
</tr>
</tbody>
</table>
Three-Schema Architecture

- Proposed to support DBMS characteristics of:
  - Program-data independence.
  - Support of multiple views of the data.

- Not explicitly used in commercial DBMS products, but has been useful in explaining database system organization.
Three-Schema Architecture

- Defines DBMS schemas at **three** levels:
  - **Internal schema** at the internal level to describe physical storage structures and access paths (e.g. indexes).
    - Typically uses a **physical** data model.
  - **Conceptual schema** at the conceptual level to describe the structure and constraints for the whole database for a community of users.
    - 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 schema.
The three-schema architecture

**Figure 2.2**
The three-schema architecture.

- **External Level**
  - External/Conceptual Mapping

- **Conceptual Level**
  - Conceptual/Internal Mapping

- **Internal Level**

- **End Users**
  - External View

- **Conceptual Schema**

- **Internal Schema**
  - Stored Database
Three-Schema Architecture

- Mappings among schema levels are needed to transform requests and data.
  - Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.
  - Data extracted from the internal DBMS level is reformatted to match the user’s external view (e.g. formatting the results of an SQL query for display in a Web page)
Data Independence

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

- **Physical Data Independence:**
  - The capacity to change the internal schema without having to change the conceptual schema.
  - 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 (continued)

- 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.
DBMS Languages

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
  - High-Level or Non-procedural Languages: These include the relational language SQL
    - May be used in a standalone way or may be embedded in a programming language
  - Low Level or Procedural Languages:
    - These must be embedded in a programming language
DBMS Languages

- **Data Definition Language (DDL):**
  - Used by the DBA and database designers to specify the conceptual schema of a database.
  - 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.
    - SDL is typically realized via DBMS commands provided to the DBA and database designers.
DBMS Languages

- **Data Manipulation Language (DML):**
  - Used to specify database retrievals and updates
  - DML commands (data sublanguage) can be *embedded* in a general-purpose programming language (host language), such as COBOL, C, C++, or Java.
    - A library of functions can also be provided to access the DBMS from a programming language
  - Alternatively, stand-alone DML commands can be applied directly (called a *query language*).
Types of DML

- **High Level or Non-procedural Language:**
  - For example, the SQL relational language
  - Are “set”-oriented and specify what data to retrieve rather than how to retrieve it.
  - Also called *declarative* languages.

- **Low Level or Procedural Language:**
  - Retrieve data one record-at-a-time;
  - Constructs such as looping are needed to retrieve multiple records, along with positioning pointers.
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.
DBMS Programming Language Interfaces

- Programmer interfaces for embedding DML in a programming language:
  - **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
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

- 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.
- Backing up the database periodically on tape.
- 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 Environments and CASE (computer-aided software engineering) tools:

- Examples:
  - PowerBuilder (Sybase)
  - JBuilder (Borland)
  - JDeveloper 10G (Oracle)
Typical DBMS Component Modules

Figure 2.3
Component modules of a DBMS and their interactions.
Centralized and Client-Server DBMS Architectures

- **Centralized DBMS:**
  - Combines everything into a 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 the 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 two-tier client server architecture

Figure 2.5
Logical 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 standard
  - JDBC: for Java programming access
- Client and server must install appropriate client module and server module software for ODBC or JDBC
- See Chapter 9
Two 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.
Three 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 for sending partially processed data between the database server and the client.
- Three-tier Architecture Can Enhance Security:
  - Database server only accessible via middle tier
  - Clients cannot directly access database server
Three-tier client-server architecture

Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.
Classification of DBMSs

- Based on the data model used
  - Traditional: Relational, Network, Hierarchical.
  - Emerging: Object-oriented, Object-relational.

- 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)
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.
History of Data Models

- Network Model
- Hierarchical Model
- Relational Model
- Object-oriented Data Models
- Object-Relational Models
History of Data Models

**Network Model:**

- The first network DBMS was implemented by Honeywell in 1964-65 (IDS System).
- Adopted heavily due to the support by CODASYL (Conference on Data Systems Languages) (CODASYL - DBTG report of 1971).
- Later implemented in a large variety of systems - IDMS (Cullinet - now Computer Associates), DMS 1100 (Unisys), IMAGE (H.P. (Hewlett-Packard)), VAX -DBMS (Digital Equipment Corp., next COMPAQ, now H.P.).
Example of Network Model Schema

Figure 2.8
The schema of Figure 2.1 in network model notation.
Network Model

- Advantages:
  - Network Model is able to model complex relationships and represents semantics of add/delete on the relationships.
  - Can handle most situations for modeling using record types and relationship types.
  - Language is navigational; uses constructs like FIND, FIND member, FIND owner, FIND NEXT within set, GET, etc.
    - Programmers can do optimal navigation through the database.
Network Model

Disadvantages:

- Navigational and procedural nature of processing
- Database contains a complex array of pointers that thread through a set of records.
  - Little scope for automated “query optimization”
History of Data Models

- **Hierarchical Data Model:**
  - Initially implemented in a joint effort by IBM and North American Rockwell around 1965. Resulted in the IMS family of systems.
  - IBM’s IMS product had (and still has) a very large customer base worldwide
  - Hierarchical model was formalized based on the IMS system
  - Other systems based on this model: System 2k (SAS inc.)
Hierarchical Model

- Advantages:
  - Simple to construct and operate
  - Corresponds to a number of natural hierarchically organized domains, e.g., organization ("org") chart
  - Language is simple:
    - Uses constructs like GET, GET UNIQUE, GET NEXT, GET NEXT WITHIN PARENT, etc.

- Disadvantages:
  - Navigational and procedural nature of processing
  - Database is visualized as a linear arrangement of records
  - Little scope for "query optimization"
History of Data Models

- **Relational Model:**
  - Proposed in 1970 by E.F. Codd (IBM), first commercial system in 1981-82.
  - Now in several commercial products (e.g. DB2, ORACLE, MS SQL Server, SYBASE, INFORMIX).
  - Several free open source implementations, e.g. MySQL, PostgreSQL
  - Currently most dominant for developing database applications.
  - SQL relational standards: SQL-89 (SQL1), SQL-92 (SQL2), SQL-99, SQL3, ...
  - Chapters 5 through 11 describe this model in detail
Object-oriented Data Models:

Several models have been proposed for implementing in a database system.

One set comprises models of persistent O-O Programming Languages such as C++ (e.g., in OBJECTSTORE or VERSANT), and Smalltalk (e.g., in GEMSTONE).

Additionally, systems like O2, ORION (at MCC - then ITASCA), IRIS (at H.P.- used in Open OODB).


Chapters 20 and 21 describe this model.
Object-Relational Models:

- Most Recent Trend. Started with Informix Universal Server.
- Relational systems incorporate concepts from object databases leading to object-relational.
- Exemplified in the latest versions of Oracle-10i, DB2, and SQL Server and other DBMSs.
- Standards included in SQL-99 and expected to be enhanced in future SQL standards.
- Chapter 22 describes this model.
Summary

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