Introductory Concepts

5DV119 — Introduction to Database Management

Umeå University
Department of Computing Science

Stephen J. Hegner
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

Three Types of Information Systems

Information-retrieval systems (IR):

- Search large bodies of information which are not specifically formatted as formal data bases.
  - Web search engine
  - Keyword search of a text base
- Typically read-only

Database management systems (DBMS):

- Relatively small schema
- Large body of homogeneous data
- Minor or no deductive capability
- Extensive formal update capability
- Shared use for both read and write

Knowledge-base systems (KBS):

- Relatively small body of heterogeneous information
- Significant deductive capability
- Typical use: support of an intelligent application

Variations of the DBMS Model

Data warehouses:

- Data are relatively static (few updates)
- Emphasis upon complex retrieval and computation

Traditional database systems with structured data:

- Geographic database systems
- Support for multimedia content
- Support for XML content

- To study these models, it is necessary to have an understanding of the basic models first.
- In this course, only traditional DBMSs will be studied.
- These variations will not be considered.

Key Issues for Database-Management Systems

Efficiency issues:

- Databases can be very large.
- Efficient access must be provided even for very large databases.

Simplicity issues:

- Many potential users are not sophisticated programmers.
  $\Rightarrow$ Simple means of access must be available.
- Complex application programs require complex access.
  $\Rightarrow$ Means of more sophisticated access must also be available.

Multi-user issues:

Concurrency: Simultaneous access to the database by several users.

Access via views: Limited "windows" through which the appropriate part of the database is viewed.

Authorization: Custom, assigned access privileges for each user.

Robustness issues:

- Deadlock and livelock must be avoided.
- A means of recovery from crashes, with minimal loss of data, must be available.

The Evolution of Data Models

Model	Development	Use	Properties	Analogy
File-management	1950s 1970s	1950s -	Low-level interaction. No data independence.	Assembly language
Navigational	1950s 1960s	1960s -	Some data independence, but the model invites dependence. Requires procedural queries.	Procedural languages
Relational	1970s -	Late 1980s -	Simple, easy to use for non-experts. Strong data independence. Standard nonprocedural query language (SQL). Excellent implementations exist. Limited expressive capability.	Declarative languages
Object-oriented	1980s -	1990s -	Powerful expressive capability, but require substantial expertise for use. Popular in niche applications. Standardization not imminent.	Object-oriented languages
Object-relational	1980s	1990s -	Attempt to integrate the simplicity of the relational model with the advanced features of the object-oriented approach. The most recent SQL standard, as well as many commercial systems, embody such features.	?
Semi-structured	1990's	2000's -	Attempt to integrate data management with markup languages, principally via XML.	?

Focus of the Course

- The course focuses upon the relational model for the following reasons:
  - The relational model is by far the most widely used.
    - It is not suitable for all applications, but there are many for which it is.
  - The relational model provides a flexible interface which has components appropriate for users at all levels.
  - A standard query language, SQL, is used with virtually all relational database systems. Thus, applications have a high degree of portability.
  - The relational model provides strong data independence: the external product is relatively independent of the internal implementation.

Multi-User Relational Database Systems

Open-Source Systems:

PostgreSQL: The most comprehensive open-source relational DBMS.

MySQL / MariaDB: Popular relational DBMSs for small systems.

- Widely used to support Web-based applications.

HyperSQL: An efficient DBMS written in Java.

- The default DBMS bundled with OpenOffice.org

SQLite: A compact DBMS written in C.

- The default DBMS bundled with LibreOffice.

The "big three" commercial relational DBMSs:

Oracle Database:

IBM DB2:

Microsoft SQL Server: (Windows only!)

Another commercial relational DBMS of interest:

Mimer SQL: Oriented towards embedded systems; based in Uppsala.

- The commercial systems listed have limited "free" versions.
- All except SQL Server run on many platforms, including Linux.
- Links may be found on the course Web page.

Single-User Relational Database Systems

Microsoft Access: The original PC DBMS for Windows.

- Part of the Microsoft Office bundle.
- It will cost you $$$, €€€, or SeKSeKSeK.
  - Even if you have a DreamSpark Premium (formerly MSDNAA) account, you cannot get MS Access for free. ☹
- Runs only under MS Windows, of course.
- Support for SQL is not as extensive as in multi-user systems.
- No real support for transactions.

Kexi: "Microsoft Access for Linux"

- Built-in SQLite-based DB server.
- Can also use other servers such as PostgreSQL and MySQL.
- Not as mature a product as MS Access.
  - ... but it is open source and free (LGPL).
- Can also be compiled for other systems.
- A link may be found on the course Web page.

Database Systems to be Used in this Course

- Both PostgreSQL and MySQL will be used as instructional systems.
- Students will receive at least one database for each on the DB servers of the department.
- If you have your own computer, each is easy to install under Linux, MS Windows, and Mac OS.
  - Some pointers for installation under Linux will be given later in the course.
- All SQL and ODBC submissions for obligatory exercises should run under both.
- You are free to (and encouraged to) try other DBMSs as well, but they will not be used in the course.

Database Access Models

SQL is the standard query language for the relational model.

- There are several access models which are built around SQL.

  Direct SQL: Write and send SQL queries directly to the database system.

  Hosting SQL within a programming language.

- Both approaches will be considered in this course.
- In the hosting approach, a framework known as ODBC (Open Database Connectivity) will be used.
  - In ODBC, special statements to communicate with databases are used in a host programming language.
  - ODBC works in principle with a variety of programming languages.
  - In this course, both *C* and *Python* will be used as host languages.
  - These languages are very different, and so the ODBC usage is quite different as well.
  - Even students who know C but not Python may find it easier to learn enough Python to use it instead of C for the exercises.

Database System Architecture

- The earliest database systems used a one-level architecture.
- The user interacted directly with the storage model.

  Analogy: assembly-language programming

Disadvantages:

- Impossible to use for non-experts.
- Difficult to use and error-prone even for experts.
- Evolution of storage model, or migration to a new architecture, requires a total rebuild of all application programs.

- For this reason, multi-level architectures were introduced and implemented.

The Two-Level Architecture

- In the *two-level architecture*, the model for internal storage is distinct from the model which the applications (users) see.

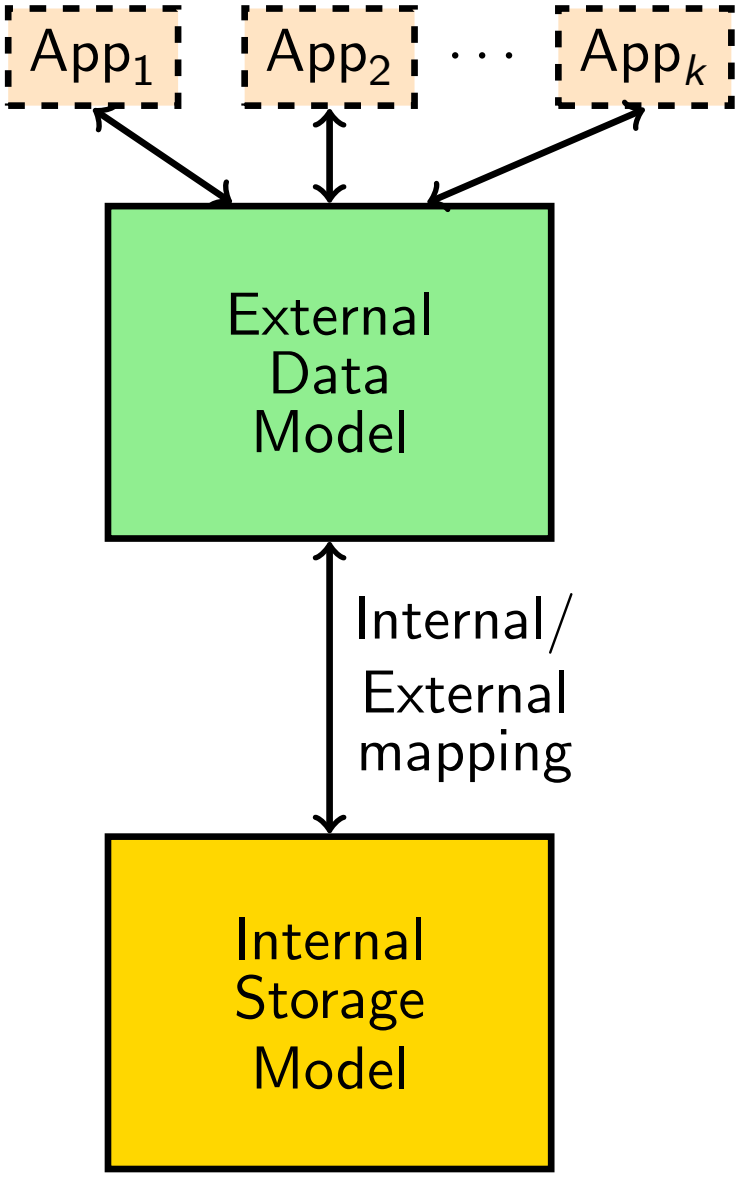
Advantages:

- The internal model and/or target architecture may be changed without requiring a rebuild of applications.

Analogy: A high-level programming language.

Disadvantages:

- All applications see a common external model.



The Three-Level Architecture

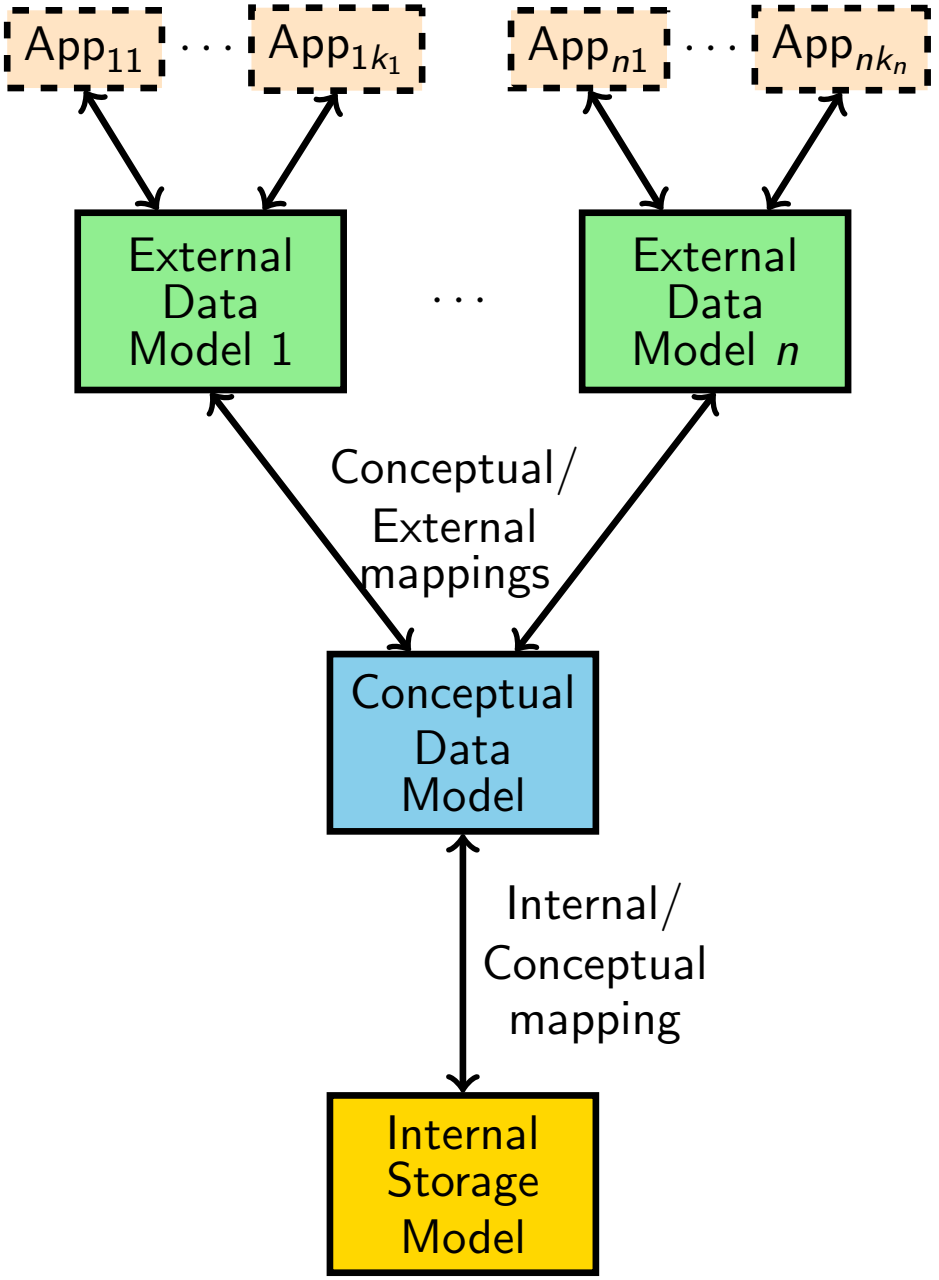
- In the *ANSI/SPARC three-level architecture*, there is an additional *conceptual model* which separates the internal and external models.

Additional advantages:

- Multiple external models may be supported.
- New external models may be introduced without requiring a new interface to the storage model.

Disadvantages:

- This model is unfortunately not seen in real systems.
- It is a design ideal.



Data Independence

Data independence refers to the idea that a more internal level of a database system may be re-engineered, or moved to a different architecture, without requiring a total rebuild of the more external layers.

- The ANSI/SPARC architecture provides two levels of data independence.
- It is often, however, something of an ideal, even with the systems of today.
- Usually, in a relational system, both the conceptual schema and the external schemata are relational.
- Still, the conceptual schema is often designed using a more general tool than the relational model.

Some Remarks on Terminology and Pronunciation

Database vs. Database (Management) System:

- A *database* is a (usually structured) collection of data.
- A *database system* or *database management system (DBMS)* is a system for managing databases.
- In the popular literature, the word *database* is sometimes used as a synonym for DBMS.
  - This terminology is confusing and its use is to be discouraged.
- MySQL and PostgreSQL are DBMSs, not databases!
- Calling a DBMS a database is akin to calling JDK a Java program.

Pronunciation of SQL:

- In research circles, it is usually pronounced as the three letters S-Q-L.
- In trade groups, SQL is sometimes pronounced as See-Quel.
  - This can lead to confusion with the older language SEQUEL, which also has that pronunciation.