Database Systems — Lecture 1 Notes: Introduction & Architecture

Sep 25, 2026 · @Haitham

Overview

This lecture answers six questions that form the theoretical foundation of the course: why databases replace shared files, what data models are, how the three-level architecture works, what data independence means, how the DBA keeps the levels separate, and what is inside a DBMS.

Main reference: Elmasri & Navathe, *Fundamentals of Database Systems*, 6th edition, Chapter 1 (Databases and Database Users) and Chapter 2 (Database System Concepts and Architecture). The running example throughout is the textbook's UNIVERSITY database.

By the end of the lecture, students should be able to:

1. Explain the problems of the file-processing approach and how the database approach solves them.
2. Define a data model, list its three categories, and distinguish schema from database state.
3. Describe the relational model using correct terms (relation, tuple, attribute, key).
4. Draw the three-schema architecture and describe each level.
5. Define logical and physical data independence and give an example of each.
6. Explain the DBA's responsibilities and the DBMS languages used at each level.
7. Name the main DBMS component modules and trace a query through them.

Part 1: Databases versus shared file systems

A database stores data once, describes it in a catalog, and shares it safely among many users; a shared file system leaves each program responsible for its own files.

Basic definitions

- **Database:** a collection of related data. The data are known facts that can be recorded and have implicit meaning. A database represents some aspect of the real world, called the *miniworld* or *universe of discourse (UoD)*. It is logically coherent and built for a specific purpose.
- **Database management system (DBMS):** a general-purpose software package that lets users *define* (specify types, structures, constraints), *construct* (store the data), *manipulate* (query, update, report) and *share* a database. Examples: MySQL, PostgreSQL, Oracle, SQL Server.
- **Database system:** the database together with the DBMS software and the application programs.
- **Meta-data:** the definition of the database, stored by the DBMS in the *catalog* (data dictionary).

The traditional file-processing approach

In file processing, each department defines and implements the files it needs for its own application. In a university, the Registrar keeps a STUDENT file, Finance keeps a FEES file, and the Library keeps a BORROWER file. Each file contains the student's name, number and address, and the record layout of each file is written inside the programs that read it.

Problem	What it looks like in the university
Data redundancy	The same address is stored in three files, wasting storage and effort.
Inconsistency	The student moves; the Registrar updates the address, Finance does not.
Program–data dependence	Adding a Birth_date field forces rewriting every program that reads the file.
Difficult access	A new question (students with unpaid fees and GPA > 3) needs a new program.
Integrity problems	Rules such as "grade must be A–F" are buried in each program's code.
Concurrent access	Two clerks update the same record at once; one update is lost.

Problem	What it looks like in the university
Security	It is hard to let Finance see balances but not grades.
Recovery	A crash in the middle of an update can leave a file half-written.

Characteristics of the database approach

In the database approach, a single repository holds data that is defined once and then accessed by many users.

1. **Self-describing nature.** The catalog holds the complete definition of the structure and constraints. The DBMS software, and users who need to know the structure, read the catalog.
2. **Insulation between programs and data (data abstraction).** The structure is stored in the catalog separately from the programs (*program–data independence*). Operations can be defined by an interface, and their implementation can change without affecting the interface (*program–operation independence*).
3. **Support of multiple views.** A view is a subset of the database, or virtual data derived from it but not stored explicitly.
4. **Sharing of data and multiuser transaction processing.** Concurrency control ensures correct results when many users update the same data. Transactions have the *isolation* property (each appears to run alone) and the *atomicity* property (all of its operations run, or none do). Systems of this kind are called online transaction processing (OLTP) applications.

Advantages of using a DBMS

- Controlling redundancy (normalization), with controlled redundancy (denormalization) when needed for performance.
- Restricting unauthorized access through a security and authorization subsystem.
- Persistent storage for program objects (object-oriented DBMSs reduce the impedance mismatch problem).
- Storage structures and search techniques for efficient queries: indexes, buffering and caching, query optimization.
- Backup and recovery.

- Multiple user interfaces, including GUIs.
- Representing complex relationships among data.
- Enforcing integrity constraints: referential integrity, key (uniqueness) constraints, business rules, inherent rules of the data model.
- Inferencing and actions through rules: deductive databases, triggers, stored procedures.
- Further implications: reduced application development time, flexibility, up-to-date information for all users, economies of scale.

When not to use a DBMS

A DBMS has costs: investment in hardware, software and training, and the overhead of providing security, concurrency control, recovery and integrity on every operation. Regular files may be better for simple applications that will not change, hard real-time systems that cannot tolerate the overhead, embedded devices with limited storage, and data that only one user accesses.

Part 2: Data models, schemas and the relational system

A data model is a collection of concepts used to describe the structure of a database, together with the operations and constraints that apply to it. It is the tool that provides data abstraction.

- **Structure:** data types, groupings of data, and relationships.
- **Operations:** basic retrievals and updates, plus user-defined operations (for example, compute a student's GPA).
- **Constraints:** rules the data must satisfy (for example, every student has a unique Student_number).

Categories of data models

Category	What it describes	Examples	Used by
High-level (conceptual)	Data as users perceive it: entities, attributes, relationships	Entity–Relationship (ER) model	Designers and users during design

Category	What it describes	Examples	Used by
Representational (implementation)	Understandable to users, yet close to how data is organized in storage	Relational, network, hierarchical, object models	Most commercial DBMSs
Low-level (physical)	How data is stored: record formats, orderings, access paths	File organizations, B+-tree and hash indexes	DBMS internals and the DBA

Self-describing data models, such as XML and key-value NoSQL systems, store the description of the data together with the data itself.

Schema versus database state

- **Database schema (intension):** the description of the database, specified during design and changed rarely. Example: STUDENT(Name, Student_number, Class, Major). A diagram of a schema is a *schema diagram*; each object in it (such as STUDENT) is a *schema construct*.
- **Database state (instance, snapshot, extension):** the data in the database at a particular moment. It changes with every insert, delete or update. Example: (Smith, 17, 1, CS), (Brown, 8, 2, CS).
- A **valid state** satisfies the structure and constraints of the schema. The DBMS stores the schema in the catalog and is responsible for keeping every state valid.
- **Schema evolution:** changing the schema after the database is in use, for example adding Birth_date to STUDENT.

The relational model and relational DBMS

The relational model was proposed by E. F. Codd at IBM in 1970. It represents all data as relations (tables) and rests on a mathematical foundation (set theory and relational algebra). It is a representational data model, and it separates the physical storage of data from its conceptual representation. This is the main reason it replaced the earlier hierarchical and network systems.

Formal term	Everyday term	UNIVERSITY example
Relation	Table	STUDENT
Tuple	Row	(Smith, 17, 1, CS)
Attribute	Column	Major
Domain	Set of allowed values	Class ∈ {1, 2, 3, 4, 5}
Primary key	Attribute that uniquely identifies a row	Student_number
Foreign key	Attribute that references another table's key	GRADE_REPORT.Student_number

A relational DBMS (RDBMS) is queried with SQL, a declarative language: the user states *what* data is wanted and the DBMS decides *how* to retrieve it. Examples: Oracle, SQL Server, MySQL, PostgreSQL, SQLite.

Example using data from Figure 1.2 of the textbook:

```
SELECT S.Name, G.Grade
FROM STUDENT S JOIN GRADE_REPORT G
  ON S.Student_number = G.Student_number;
```

Part 3: The three-level (three-schema) architecture

The three-schema architecture, proposed by the ANSI/SPARC study group, separates user applications from the physical database by describing the database at three levels, each with its own schema.

```
flowchart TB
  V1[External view 1] <--> C[Conceptual schema]
  V2[External view 2] <--> C
  V3[External view 3] <--> C
  C <-->|conceptual/internal mapping| I[Internal schema]
  I <--> D[(Stored database)]
```

The arrows between the external views and the conceptual schema form the external/conceptual mapping; the arrow between the conceptual and internal schemas is the conceptual/internal mapping.

Characteristics of each level

Level	Its schema describes	Hides	Data model used	UNIVERSITY example
External (view)	Only the part of the database one user group needs	Everything that group does not need	Representational (often relational views)	Advisor view: student name, class and transcript only
Conceptual	The whole database for all users: entities, data types, relationships, constraints, user operations	Physical storage details	Representational, often designed first with a high-level (ER) model	Tables STUDENT, COURSE, SECTION, GRADE_REPORT, PREREQUISITE with their keys
Internal	Physical storage: record layouts, file organization, indexes, access paths	Nothing below it	Physical data model	STUDENT stored as a file with a B+-tree index on Student_number

There are many external schemas (one per user group), but only one conceptual schema and one internal schema per database.

Mappings

The three schemas are only descriptions; the only data that really exists is at the physical level. The DBMS must therefore transform a request expressed on an external schema into a request on the conceptual schema, and then into a request on the internal schema over the stored database. Retrieved data must be reformatted back up the levels to match the user's view. These transformations are called **mappings**, and their definitions are kept in the catalog.

Why the architecture is necessary

- **Different users, different needs:** each user group gets a tailored view of the same shared data.
- **Change without breaking:** storage or structure can change at one level while programs at other levels keep working. This is data independence (Part 4).
- **Security and simplicity:** users see only what they are authorized to see, and they never deal with storage details.

The price is run-time overhead: every request and result passes through the mappings. For this reason many DBMSs do not separate the three levels completely, but the architecture remains the model for explaining how independence is achieved.

Part 4: Data dependence versus data independence

Data independence is the capacity to change the schema at one level without having to change the schema at the next higher level. Data dependence is its opposite, and it is the normal situation in file systems.

Data dependence in file systems

In a file system, the record layout is hard-coded in every program that reads the file:

```
/* repeated inside every program */
struct Student {
  char name[30];
  char number[4];
  int  class;
  char major[4];
};
fread(&s, sizeof(struct Student), 1, fp);
```

If the Registrar adds a Birth_date field, widens Name, or reorders fields, every program that reads the file must be edited and recompiled, even programs that never use the changed field. Programs depend on how the data is stored.

Two kinds of data independence

	Logical data independence	Physical data independence
Definition	Change the conceptual schema without changing external schemas or application programs	Change the internal schema without changing the conceptual schema (and therefore the external schemas)
Example changes	Add a table or attribute (Birth_date to STUDENT); remove an attribute no view uses; change constraints	Add an index on Major; reorganize files; change record placement; move data to a faster disk
What changes instead	Only the external/conceptual mapping	Only the conceptual/internal mapping
Difficulty	Harder: programs depend closely on data structure and constraints	Easier: supported by most DBMSs

How the DBMS addresses both issues

1. The structure of the data lives in the catalog, not in the programs.
2. When a schema changes at one level, the DBA updates the mapping between that level and the level above it.
3. Views and programs at the higher level remain unchanged; only the mapping absorbs the change.

Situation	File system	DBMS
Where the structure lives	Inside each program	In the system catalog
Adding a field	Rewrite every program	Alter the schema; update the mapping
Adding an index	Rewrite the access code	CREATE INDEX; queries stay the same

This is why the self-describing catalog (Part 1) and the three-level architecture (Part 3) together solve the data dependence problem.

Part 5: The DBA's role and DBMS languages

The database administrator (DBA) is the person, or team, who establishes and maintains the separation of the three levels: the DBA defines the schema at each level and keeps the mappings between them correct.

The DBA's responsibilities

Responsibility	Level it relates to
Defines the conceptual schema (together with database designers) using the DDL	Conceptual
Defines the internal schema: storage structures, indexes, file placement	Internal
Defines external schemas: creates views for each user group	External
Maintains the mappings so that a change at one level stays inside that level	Between levels
Authorizes access: grants each user only the views and privileges needed	External
Monitors and tunes performance: adds indexes, reorganizes storage, invisibly to users	Internal
Acquires hardware and software resources; coordinates and monitors database use	Whole system

Other people involved with the database (Chapter 1): database designers identify the data to be stored and choose structures for it; end users (casual, naive/parametric, sophisticated, standalone) use the database; system analysts and application programmers specify and build the applications. Behind the scene, DBMS designers and implementers build the DBMS, tool developers build tools, and operators and maintenance personnel run the hardware and software.

DBMS languages at each level

Language	Purpose	Level	SQL example
DDL (data definition language)	Define the conceptual schema	Conceptual	CREATE TABLE STUDENT (…)
SDL (storage definition language)	Define the internal schema	Internal	CREATE INDEX idx_major ON STUDENT(Major)
VDL (view definition language)	Define user views and their mappings	External	CREATE VIEW Advisor_view AS SELECT …
DML (data manipulation language)	Retrieve, insert, delete, update data	All users	SELECT, INSERT, UPDATE, DELETE

In modern relational DBMSs, SQL is a single comprehensive language that combines DDL, VDL and DML. Storage definitions are mostly DBA-controlled parameters; CREATE INDEX is not part of the SQL standard, but all major DBMSs support it.

DMLs come in two kinds:

- **High-level (nonprocedural, set-at-a-time):** specifies *which* data to retrieve, e.g. SQL. Can be used interactively or embedded in a host language.
- **Low-level (procedural, record-at-a-time):** retrieves individual records and processes them in loops; must be embedded in a general-purpose programming language.

Part 6: DBMS components and how they work together

A DBMS is a set of cooperating modules: compilers and an optimizer turn user requests into executable code, a runtime processor executes it, and storage, catalog, concurrency and recovery modules keep the data correct and available (textbook Figure 2.3).

```
flowchart TB
  DBA[DBA staff] --> DDL[DDL compiler]
  CU[Casual users] --> QC[Query compiler]
  AP[Application programmers] --> PC[Precompiler + DML compiler]
  PU[Parametric users] --> CT[Compiled transactions]
  DDL --> CAT[(System catalog / data dictionary)]
  QC --> QO[Query optimizer]
  PC --> HC[Host language compiler]
  CAT -.-> QO
  QO --> RDP[Runtime database processor]
  HC --> RDP
  CT --> RDP
  CAT -.-> RDP
  RDP --> SDM[Stored data manager]
  RDP --> CC[Concurrency control, backup and recovery]
  SDM --> DB[(Stored database)]
```

The component modules

Module	Role
DDL compiler	Processes schema definitions and stores the descriptions (meta-data) in the catalog
System catalog / data dictionary	Holds names, sizes and types of data items, storage details, mappings and constraints; consulted by most other modules
Interactive query interface and query compiler	Accepts ad hoc queries from casual users, parses and validates them against the catalog
Query optimizer	Rearranges and reorders operations and chooses efficient algorithms and indexes; produces executable code
Precompiler	Extracts DML commands from application programs written in a host language (C, Java, etc.)
DML compiler and host language compiler	Compile the extracted DML and the rest of the program; linked into canned (compiled) transactions

Module	Role
Runtime database processor	Executes privileged commands, executable query plans and canned transactions
Stored data manager	Controls access to data on disk through the operating system; manages main-memory buffers
Concurrency control, backup and recovery subsystems	Integrated with the runtime processor for transaction management

How they work together: one query's journey

1. A casual user types a SQL query in the interactive query interface.
2. The query compiler parses it and checks names and types against the system catalog.
3. The query optimizer chooses an efficient execution plan, for example using the index on Student_number.
4. The runtime database processor executes the plan.
5. The stored data manager brings the needed disk pages into memory buffers.
6. Concurrency control and recovery keep the result correct despite other users and failures; the answer returns to the user.

For an application program, the precompiler separates the SQL, the DML compiler compiles it, and the linked result becomes a canned transaction that parametric users run repeatedly.

Utilities, interfaces and tools

- **Utilities (mainly for the DBA):** loading data from files, backup, database storage reorganization, performance monitoring, sorting, compression.
- **User interfaces:** menu-based, forms-based, graphical (GUI), natural-language and keyword search, interfaces for parametric users, privileged interfaces for the DBA.
- **Tools and environment:** CASE tools for database design, data dictionary (repository) systems, application development environments, communication and client–server software.

Summary and review questions

Question	Answer in one line
Files vs. database	A DBMS stores data once, describes it in a catalog, and shares it safely among many users.
Data models	Concepts for structure, operations and constraints; relational = tables, keys and SQL.
Three levels	External views, one conceptual schema and one internal schema, linked by mappings.
Independence	Change one level and only the mapping changes: logical and physical data independence.
The DBA	Defines the schema at each level, maintains the mappings, controls access, tunes performance.
The DBMS	Compilers, optimizer, runtime processor, stored data manager, catalog, concurrency and recovery.

Review questions:

1. List four problems of the file-processing approach and explain how a DBMS solves each.
2. Distinguish between a schema and a database state using the UNIVERSITY example.
3. Name the three categories of data models and give one example of each.
4. Draw the three-schema architecture and label both mappings.
5. Give one change that needs logical and one that needs physical data independence. Which is harder to achieve, and why?
6. State three responsibilities of the DBA and link each to a level of the architecture.
7. Trace a SELECT query through the DBMS component modules.

Reading: Elmasri & Navathe, 6th edition, Chapters 1 and 2.