Course objectives — Advanced Databases

Objectives

Provide an **advanced-level training** (Master's Year 1, 2025) that extends the foundational **Database Systems** module, with the goal of strengthening mastery of modern data systems.

The course aims to :

- **Reinforce** core knowledge : relational model, transactions, and DBMS architecture ;
- **Master advanced SQL** : subqueries, joins, functions, views, *triggers*, procedures, and common table expressions (CTE) ;
- **Introduce** the **object–relational** paradigm (SQL3) : complex types, inheritance ;
- **Explore NoSQL**, **distributed**, and **cloud-based** approaches ;
- **Develop** skills in database design, administration, and optimization (performance and security).

Final goal : combine **theoretical rigor** with **applied practice**.

Prerequisites

Basic knowledge of **relational databases**, **SQL**, and **relational algebra**.

Course motivations

Why this course ?

Data is the **fuel of the digital economy** and a central pillar of **data science**. Its value creation relies on four key stages :

1. **Collection** : sensors, logs, files or databases, REST APIs ;
2. **Preparation** : data cleaning, integration, and structuring ;
3. **Analysis** : statistics, machine learning, and data mining ;
4. **Value creation** : decision support, artificial intelligence, and service delivery.

Practical challenges

Data is pervasive across **healthcare**, **finance**, the **cloud**, and **social networks**. Modern systems require :

- **Reliability** — ensuring data integrity and preventing information loss ;
- **Scalability** — handling massive data volumes and millions of concurrent users ;
- **Resilience** — maintaining service continuity in the face of failures.

These requirements lie at the core of today's major technological challenges : **Big Data**, **Cloud Computing**, and **Artificial Intelligence**.

Technologies studied (Open-source ecosystem)

PostgreSQL

- A **relational and object–relational** DBMS.
- Full support for **advanced SQL** : views, functions, transactions.
- Ensures **robustness**, **consistency**, and **data integrity**.
- Extensible through modules such as `PostGIS (Geographic Information System)`.
- Widely used in **mission-critical and industrial** applications.

MongoDB

- A **NoSQL** document-oriented database.
- Stores data as rich **JSON documents**.
- Offers high **schema flexibility**.
- Provides **horizontal scalability** through replication and sharding.
- Well-suited for **Big Data** and modern **web applications**.

Chapter 1

Introduction to Advanced Databases

Mr. Laïdi FOUGHALI

l.foughali@univ-skikda.dz

(Course materials ⇒ al-moualime.com)

University of Skikda — Department of Computer Science

1st Year Master RSD/AI

Advanced Databases (ADB)

October 1/14/21/29, 2025

Version 1.0 (Initial) — 2025-10-25 à 07:54:13

Outline

1 Introduction

2 Basic concepts

3 Advanced concepts

4 Architectures

5 Security

6 Conclusion

In the era of digital data

Today, **digital data** is ubiquitous and shapes our daily lives. It is :

- generated by **smartphones** (communications, mobility, social interactions),
- produced by **smartwatches** (health, activity, sleep),
- emitted by **smart objects** (cars, homes, voice assistants),
- collected even on **pets** (GPS trackers).

Scientific and technological stakes

⇒ Data production is **massive**, **varied**, and **continuous**.

⇒ It requires tools to make data **persistent** so it can be better **exploited** and **monetized**.

Global data volume

[Rivery, 2024]

2024 : **149 ZB** (1 ZB $= 10^{21}$ bytes $\approx 10^{12}$ hard drives of 1 TB).

2025 (projection) : **181 ZB**.

Big Data

What is Big Data ?

Big Data refers to **massive**, varied, high-velocity data that traditional systems can no longer process efficiently.
It is characterized by the "3Vs" :

- **Volume** : huge quantities generated continuously,
- **Velocity** : production and processing in real time,
- **Variety** : multiple formats (text, images, videos, sensors).

Main challenges :

- collect and organize this massive flow while limiting inconsistencies and redundancy,
- exploit data to create **added value** (decision, prediction, automation, **AI**).

Big Data market *[Grand View Research, 2024]*

2024 : **$373.9B** **2030 (forecast)** : **$862.3B** (growth : **14.9%**).

Why use databases (DBs) ?

To handle the information explosion, organizations build *data pipelines* : collect $\rightarrow$ transform $\rightarrow$ store $\rightarrow$ analyze. In this context, **Database Management Systems (DBMSs)** play a central role. They provide :

- **structured and durable** storage,
- **security and consistency** of information,
- **fast and controlled** access.

Use cases

Services such as **iCloud** (Apple) and **Google Drive**, as well as many **industrial** and **financial** sectors, rely on **databases**.

Database market *[RCR Wireless ; TechBlog, 2023]*

2023 : **$805.7B** (+10.6% vs 2022).
Forecast 2023–2027 : **+10.4%/yr**.

Database use case : iCloud (Apple)

What is iCloud ?

iCloud is Apple's **cloud storage and synchronization** service. It lets you **back up** devices (iPhone, iPad) and **keep photos**, **videos**, **contacts**, **calendars**, and **documents** up to date across **all devices** (iPhone, iPad, Mac) and on the **Web** (`icloud.com`).

Technically, iCloud relies on :

- **data centers** distributed worldwide for availability,
- **relational databases** (contacts, calendars) and **NoSQL stores** (photos, videos),
- **strong security** : data encryption and Apple ID identity.

Conceptual point

iCloud shows that **databases** are essential to run a service used by millions of people every day.

[Apple — iCloud Overview]

Diversification of databases

Since the 1970s—with the emergence of the **relational model**—DBMSs have evolved to meet diverse needs :

- **Relational databases (SQL)** : robustness, integrity (constraints, ACID transactions), and normalization (e.g., banking transactions).
- **NoSQL databases** : flexibility and scalability for massive volumes (e.g., social networks, **Big Data**).
- **Real-time applications** : high performance for instantaneous streams (e.g., finance, messaging).
- **Distributed and cloud systems** : management of colossal volumes by major players (**Google**, **Apple**, **Meta**, **Amazon**, **Microsoft**).

Link with AI

AI models such as **ChatGPT** and **DeepSeek** are trained on immense text and multimedia corpora. In practice, one often combines :

- **Relational databases** for **metadata** (users, experiments, traceability) ;
- **NoSQL / distributed systems** and **object storage** for **massive corpora** and **logs**.

Conclusion

Key message

Databases are a **cornerstone of digital systems**, at the heart of **everyday**, **industrial**, **scientific**, and **artificial intelligence** use cases.

Basic concepts of databases [1/10]

Data

Raw, coded fact with no particular meaning when taken in isolation.

Examples : 37 ; "Skikda" ; "2025-09-28".

Information

Data **interpreted by a user (or an application)** within a given **context**, so that it acquires meaning relative to a need.

Examples : 37 = a person's age ; "Skikda" = city of birth.

Metadata

Data that describes other data (catalogs, dictionaries, schemas).

Example : table `Client` with column `Nom` (text type, length 50).
Note : the DBMS manages storage, integrity, and access ; it does not "understand" the meaning of data.

[Silberschatz, Korth & Sudarshan (2020)]

Basic concepts of databases [2/10]

Schema

Structural and persistent representation of the database. It describes the **logical objects** (tables, columns, types, constraints, views) and their relationships. The schema serves as a **master plan** from which data are organized.

Example : a table `Etudiant(NumEtudiant, Nom, DateNaissance, Filiere)` is part of the university's logical schema.

Instance

The set of **actual data** present in the database at a given time. The instance reflects the **current state** of the database and **evolves over time**, unlike the schema, which typically remains stable.

Example : the rows corresponding to students enrolled this year constitute an instance of the preceding schema.

[Silberschatz, Korth & Sudarshan (2020)]

Basic concepts of databases [3/10]

Entity

Real or conceptual object modeled in a database.

Example : a student.

Attribute

Property characterizing an entity or a relationship.

Examples : `Nom`, `DateNaissance`.

Identifier (primary key)

Attribute or combination of attributes that uniquely identifies each entity.

Association

Conceptual link between entities in a conceptual data model.

[Silberschatz, Korth & Sudarshan (2020)]

Basic concepts of databases [4/10]

Relation

Logical table composed of a set of columns (attributes) and rows (tuples).
Each row represents an occurrence of an entity or an association.

Predicate

Logical statement that each tuple makes true when it belongs to the relation.

Integrity constraints

Rules that ensure the validity and consistency of data :

- **Entity** : uniqueness of an identifier.
- **Referential** : correspondence between primary and foreign keys.
- **Business/domain** : compliance with application-defined conditions.

[Codd (1970) ; Silberschatz, Korth & Sudarshan (2020)]

Basic concepts of databases [5/10]

Association vs Relation

- An **association** links several entities in the conceptual model (E/R).
- A **relation** represents a table in the relational model.

When moving from the conceptual model to the relational model, each association becomes a relation.

Example : in the conceptual model, a student enrolls in a course ; in the relational model, this becomes table `INSCRIPTION(NumEtudiant, CodeCours, Date)`.

[Chen (1976) ; Codd (1970)]

Basic concepts of databases [6/10]

Relational model *[Codd, 1970]*

Foundation of modern database systems :

- Data organized in **tables**.
- **Rows** (tuples) and **columns** (attributes).
- **Keys** : primary and foreign, for consistency.
- **Normalization** to reduce redundancy.

Associated formalisms

- **Relational algebra** : selection, projection, join, union, etc.
- **Relational calculus** : declarative approach based on predicate logic.

Basic concepts of databases [7/10]

Database

Structured and coherent set of information, organized to be :

1. **stored** durably,
2. **updated** efficiently and without inconsistencies,
3. **queried** quickly and securely.

A relational database relies on the **relational model**.

Example : a university system stores students, courses, and enrollments in a relational database.

Analogy (library)

- **Books** = Data **Shelves** = Structures (tables, indexes)
- **Readers** = Users **Building** = Physical support (server)

The library illustrates the distinction between **content** (data) and its **physical and logical organization** (structures managed by the DBMS).

[Codd (1970) ; Silberschatz, Korth & Sudarshan (2020)]

Basic concepts of databases [8/10]

Transaction

Set of operations executed as a **single logical unit of work**. A transaction guarantees the **ACID** properties :

- **Atomicity** : all operations execute entirely or none at all.
- **Consistency** : each transaction preserves database validity (constraints satisfied).
- **Isolation** : concurrent transactions do not interfere.
- **Durability** : once committed, changes persist despite failures.

Example : a bank transfer where the debit of account A and the credit of account B are executed together or rolled back together.

[Silberschatz, Korth & Sudarshan (2020)]

Basic concepts of databases [9/10]

Index

Auxiliary data structure that speeds up access to table rows. It maps values of one or more columns to the physical location of matching records. Without an index, the DBMS must scan the entire table to locate data.

Index types

- **B-tree** : suited to lookups, ordering, and ranges (most common).
- **Hash** : efficient for equality comparisons.
- **Bitmap** : useful for columns with low cardinality.
- **Full-text** : designed for word search in text.

Remark

Indexes improve read performance, but their maintenance can slow updates and inserts. They are part of the physical structures managed by the DBMS.

[Silberschatz, Korth & Sudarshan (2020)]

Basic concepts of databases [10/10]

Database Management System (DBMS) *[Elmasri & Navathe, 2017]*

Software that **creates**, **manipulates**, and **administers** a database, hiding hardware and technical details. Modern DBMSs rely on the **relational model** (**ISO/IEC 9075**).

Main functions

- **Define** structure : tables, views, constraints.
- **Manipulate and query** : insert, delete, update, search.
- **Secure** access : authentication, roles, permissions.
- **Optimize** : execution plans, caching, indexing.

Key strengths

Data integrity and consistency ; **Transactional reliability (ACID)** ; **Standardization (SQL, ISO/IEC 9075)** ; **Mature ecosystem** : PostgreSQL, Firebird, MySQL, Oracle, SQL Server.

Structured Query Language (SQL)

SQL *[ISO/IEC 9075]*

Standardized language to **define**, **manipulate**, **control**, and **secure** data in a relational database.

Sublanguages :

- **DDL** (*Data Definition Language*) — Define structure ⇒ `CREATE`, `ALTER`, `DROP`.
- **DML** (*Data Manipulation Language*) — Manipulate content ⇒ `SELECT`, `INSERT`, `UPDATE`, `DELETE`.
- **DCL** (*Data Control Language*) — Manage access rights ⇒ `GRANT`, `REVOKE`.
- **TCL** (*Transaction Control Language*) — Control transactions ⇒ `COMMIT`, `ROLLBACK`, `SAVEPOINT`.

Historical database models

Historical milestones

- **Hierarchical model** (IBM IMS, 1960s) : data organized as **trees** (parent $\rightarrow$ children). *Example :* a customer owns several orders.
- **Network model** (CODASYL, 1970s) : data linked as **graphs** via pointers. *Example :* a student is linked to multiple courses.

Key takeaway

These models are **ancestors of the relational model** : very performant but rigid, they paved the way for Codd's relational model (1970).

Limits of relational databases

Relational systems have several limitations in modern contexts :

- Difficulty handling **massive data volumes** or **unstructured data** (images, videos, social streams).
- Limited **horizontal scalability** : efficiently distributing data across many servers is complex.
- Less suited to applications requiring **real-time processing** or involving **highly complex relationships** (graphs, social networks, etc.).

Consequence

These limits have led to the emergence of so-called **NoSQL** databases, designed for specific needs :

- **MongoDB** — *document-oriented model* : stores semi-structured data (JSON, web apps).
- **Cassandra** — *column-oriented model* : high availability and scalability.
- **Neo4j** — *graph model* : representation of complex networks (social graphs, semantic links).

NoSQL : another way to manage data

Definition *[Sadalage & Fowler, 2013]*

The term **NoSQL** means *"Not Only SQL"*. It refers to a group of data management systems that provide :

- **Schema flexibility** — structures can evolve freely ;
- **Horizontal scalability** — data can be distributed across multiple servers ;
- High performance for specific use cases (real time, massive data, etc.).

Main families and use cases :

- **Key–Value** — (Redis, DynamoDB) : ultra-fast storage, caching, user sessions.
- **Document** — (MongoDB, CouchDB) : semi-structured data (JSON/BSON), web applications.
- **Graph** — (Neo4j, OrientDB) : modeling complex networks (relations, social graphs).

Distributed databases

Definition *[Özsu & Valduriez, 2020]*

A **distributed database** stores data across several **interconnected nodes**, yet is managed as a single **DBMS**. Goals : **availability**, **fault tolerance**, **performance**.

Core mechanisms

- **Fragmentation** : dividing data into logical subsets.
- **Replication** : duplicating fragments for reliability and fast access.
- **Transparency** : users query the database without knowing the actual data location.
- **Consistency** : synchronization across replicas (strong, eventual, or hybrid).

Examples of transparency :

- **Google Spanner** : a single global database visible to the user, even though it is distributed across continents.
- **Amazon DynamoDB** : automatic distribution and replication, without manual placement management.

Object-oriented databases (OODB)

Definition

An **object-oriented database** (OODB) directly stores **objects** from object-oriented programming—both their **data** and their **methods**.

Main characteristics

- **Encapsulation** : data and methods are grouped inside the same object.
- **Inheritance** : a class can reuse and extend another's properties.
- **Polymorphism** : one interface can have multiple behaviors.
- **Object identity** : each object retains a unique identity, independent of its values.

Exemple orienté objet sous PostgreSQL

Principle

PostgreSQL enables an **object–relational** approach : **composite types** describe objects, and **functions** act as methods.

```
CREATE TYPE voiture AS (
  marque        text,
  couleur       text,
  prix          numeric,
  numero_serie  text
);

CREATE OR REPLACE FUNCTION taxe(v voiture) RETURNS numeric AS $$
  SELECT v.prix * 0.02;
$$ LANGUAGE sql IMMUTABLE;

SELECT taxe(ROW('Toyota','rouge',20000,'ABC123')::voiture);
```

Key idea

PostgreSQL makes it possible to associate **functions** with **composite types**, providing genuine **object-oriented behavior**.

Business constraints in modern DBMSs

Modern DBMSs (such as **PostgreSQL** or **Firebird**) go beyond traditional constraints (**PRIMARY KEY**, **FOREIGN KEY**, **CHECK**, **DOMAIN**) : they allow **centralizing business logic on the server** to ensure data consistency and reliability.

Complementary mechanisms :

- **Triggers** : automatically execute an action on `INSERT`, `UPDATE`, or `DELETE`.
- **Stored procedures** : encapsulate business logic within the DBMS for server-side reuse.
- **Views** : simplify data access and strengthen security.

Pedagogical note

⇒ **PostgreSQL** : full-featured, extensible DBMS widely used in teaching and research.

⇒ **Firebird** : lightweight and robust DBMS, ideal for practice and learning.

Specialized databases

Examples of specialization

- **Text-oriented databases** : store **raw files** (system logs, configurations).
  ⇒ Simple and fast for writes, but limited for complex queries.
- **Multimedia databases** : designed to manage **images**, **videos**, **audio**, with adapted indexing.
  ⇒ Used in web platforms, streaming, and AI (vision, speech recognition).
- **Autonomous databases** : integrate **AI** for self-management (optimization, security, backup).
  ⇒ Example : **Oracle Autonomous Database**.

Key takeaway

These databases address **specific needs** (text, multimedia, self-management). They do not replace relational or NoSQL databases but rather **complement** them in specialized contexts.

The ANSI/SPARC model

Definition *[Elmasri & Navathe, 2016]*

The **ANSI/SPARC model**, proposed in **1975** by the **ANSI** (American National Standards Institute) and **SPARC** (Standards Planning and Requirements Committee), defines a **three-level architecture** for structuring a **DBMS**.

The three levels

- **Internal (physical) level** : describes actual storage (*files, blocks, indexes*).
  ⇒ *Goal :* optimize performance and hardware resource usage.
- **Conceptual (global logical) level** : provides an abstract, unified view independent of storage (*tables, relations, constraints*).
  ⇒ *Goal :* ensure consistency and logical independence.
- **External (user views) level** : offers representations tailored to user or application needs.
  ⇒ *Goal :* simplify access and enhance security.

Client–server architecture

General principle *[Silberschatz, Korth & Sudarshan, 2020]*

The **client–server** architecture relies on network communication via **IP sockets** :

- The **client** (e.g., *psql*, *pgAdmin*, application) sends SQL queries to the server.
- The **server** (e.g., *PostgreSQL*, *Firebird*) listens on an IP address and port, processes queries, and returns results.

Operation *[Silberschatz, Korth & Sudarshan, 2020]*

1. Open a network connection (TCP/IP socket).
2. Exchange requests and responses, typically in SQL.
3. Close the socket at the end of the session.

Advantages

Standardized communication, secure remote access, and support for multiple simultaneous clients.

Distributed architectures

Principle

When a single server is no longer sufficient, the DBMS can be deployed across several interconnected sites or nodes : a **distributed architecture**. This setup balances load, improves availability, and facilitates scaling.

Architecture forms

- **Fragmented** : each site manages a distinct subset of data (e.g., customers by region).
- **Replicated** : each site keeps a full or partial copy of the database.
- **Hybrid** : a combination of both strategies depending on access and performance needs.

Advantages

Fault tolerance, horizontal scalability, data proximity to users, and service continuity even if a node fails.

[Özsu & Valduriez, 2020]

Local versus cloud deployment

On-premises databases *[DataBird, 2025]*

Hosted on the organization's internal servers.

- **Advantage** : full control over configuration, security, and data.
- **Drawbacks** : high maintenance, backup, and administration costs, and limited accessibility outside local infrastructure.

Cloud-hosted databases (DBaaS : *Database as a Service*)

Hosted by providers such as **Amazon**, **Google**, or **Microsoft**.

- Global accessibility and an **on-demand** model (*pay-per-use*).
- Automatic backups, updates, and replication.
- High performance and near-unlimited scalability.
- Reduced internal management and maintenance costs.

$\Rightarrow$ **Cloud computing** is now the preferred solution for most companies.

Open-source databases

Definition *[DataBird, 2025]*

An **open-source database** is a management system whose **source code is freely accessible, modifiable, and redistributable** under an open license.

It is notable for its **flexibility**, **transparency**, and **no license fees**, with **continuous improvement** driven by an active community.

Main limitations include predominantly **community-based support**, potentially **complex administration**, and **variable documentation** by project.

Representative examples

- **PostgreSQL** — standards-compliant SQL, known for robustness and extensibility.
- **MongoDB (Community Edition)** — well-suited to semi-structured data and modern web applications.

MVCC : Multiversion Concurrency Control

In addition to **transactions**, modern DBMSs use **MVCC (Multiversion Concurrency Control)** to manage concurrency. It maintains **multiple versions of the same data** to allow parallel execution that is both **fast** and **consistent**.

Principle :

- Each transaction accesses a **snapshot** of the database at start time.
- When data is modified, a **new version** is created without deleting the old one.
- Readers see a **stable and consistent** database, while writers produce new versions.

Consequences :

- Reads do not block writes → **high parallelism**.
- Each transaction remains isolated and consistent.
- Old versions are later **cleaned up** (**VACUUM**, **garbage collector**).

Concurrency : problems and solutions

When multiple transactions run in parallel, several anomalies can occur :

- **Dirty Read** : reading modified but uncommitted data.
- **Lost Update** : one update is overwritten by another transaction.
- **Non-repeatable Read** : a value changes between two successive reads.
- **Phantom Read** : rows appear or disappear between identical queries.
- **Deadlock** : circular wait where two transactions wait on each other.

Control mechanisms :

- **2PL (Two-Phase Locking)** : acquire then release locks in strict order $\Rightarrow$ guarantees **serializability**.
- **MVCC (Multiversion Concurrency Control)** : each transaction accesses a **consistent snapshot** $\Rightarrow$ enables non-blocking read parallelism.
- **Deadlock detection** : the DBMS detects cycles and aborts one transaction via `ROLLBACK`.

Crash recovery

Beyond concurrency control, another major challenge is **crash recovery**. Failures (power outages, software crashes, disk faults) are inevitable. The DBMS must be able to :

- restore a **consistent state** of the database ;
- preserve **committed transactions** ;
- roll back those that were **interrupted** ;
- ensure a **fast restart** of service.

Recovery mechanisms :

- **WAL (Write-Ahead Logging)** : log operations before they are written.
- **Checkpoints** : record intermediate recovery points.
- **Redo / Undo** : replay committed transactions and undo incomplete ones.
- **Regular backups** : limit data loss in case of disaster.
- **DRP (Disaster Recovery Plan)** : ensure continuity after a major outage.

Reliability and transactional security

Goal

Ensure that database operations are **reliable**, **consistent**, and **recoverable**, even under failures or concurrency.

Mechanism	Main role	Main benefit
2PL	Lock control to prevent transaction conflicts.	**Serializability**
MVCC	Simultaneous access via multiple data versions.	**High parallelism**
WAL	Log operations beforehand.	**Reliable recovery**
Checkpoints	Periodically save a consistent state.	**Fast recovery**
DRP	Plan recovery after major disasters.	**Service continuity**

These mechanisms are the pillars of DBMS reliability.

Conclusion (chapter summary)

- **Data** is the **core of modern digital systems** (it powers services, AI, and decision-making).
- **Models** have diversified from **relational** to **NoSQL** via the **object–relational** paradigm.
- **Architectures** evolved from the **ANSI/SPARC** model to **distributed and cloud** systems.
- **Security** relies on **transactional reliability**.

Key figures and final message

- Global data volume : **149 ZB (2024)** $\rightarrow$ **181 ZB (2025)**.
- Global market estimated at **$862.3 billion by 2030**.

$\Rightarrow$ **Databases** are the **invisible foundation of the digital world**.

$\Rightarrow$ Their **modeling**, **architecture**, and **security** determine the **reliability**, **performance**, and **value** of modern systems.