# Objectives of the "Advanced DataBases" course

## Objectives

Provide **advanced** training (Master 1) extending the **databases (DB)** module in order to strengthen the mastery of modern systems.

The aim is to:

- **Consolidate** the fundamentals: 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;
- **Open up** to **NoSQL**, **distributed** and **cloud** approaches;
- **Develop** skills in design, administration and optimization (performance and security).

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

## Prerequisites

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

# Motivations for the course

## Why this course?

Data is the **fuel of the digital economy** and a central pillar of **data science**. Turning it into value relies on four fundamental steps:

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

## Practical stakes

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

- **Reliability** — guaranteeing integrity and no loss of information;
- **Scalability** — handling massive volumes and millions of concurrent users;
- **Resilience** — ensuring continuity of service despite failures.

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

# Technologies to be studied (open-source ecosystem)

## PostgreSQL

- **Relational and object–relational** DBMS.
- Full support for **advanced SQL**: views, functions, transactions.
- Guarantees data **robustness**, **consistency** and **integrity**.
- Extensible with modules such as `PostGIS (Geographic Information System)`.
- Used in many **mission-critical and industrial applications**.

## MongoDB

- Document-oriented **NoSQL** database.
- Storage in the form of **enriched JSON documents**.
- Offers great **schema flexibility**.
- Enables **horizontal scalability** (replication and sharding).
- Suited to **Big Data** and **modern web applications**.

# Chapter 1

## Introduction to advanced databases

**Mr Laïdi FOUGHALI**

l.foughali@univ-skikda.dz

(Website ⇒ www.al-moualime.com)




**University of Skikda — Department of Computer Science**

1st Year Master NDS/AI

Advanced DataBases (ADB)

***Updated on 2026-10-30 at 10:07:54***




© 2026 Mr Laïdi FOUGHALI — www.al-moualime.com/adb

Creative Commons License — BY-NC-SA 4.0 International

# Course conventions

Each page has a **colour** that shows its role:

**Blue**: the lecture, to read and to run.

**Orange**: an exercise or a recall; look for the answer.

**Green**: the solution of the exercise, on the next page.

- **Method**: answer the exercise **before** turning the page.
- **Review**: this link takes you to the lecture page that holds the answer; the **"↩ Back to slide N"** button (or the R key) brings you back to the exercise (in a PDF: Alt+←).
- **Interactive version**: the **Run** button runs the SQL example in the browser; "Reset database" restores the initial data.

# Outline

1. Introduction
2. Basic concepts
3. Advanced concepts
4. Architectures
5. Security and reliability
6. Conclusion

# In the age of digital data

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

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

## Scientific and technological stakes

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

⇒ It requires tools to make data **persistent**, in order to better **exploit** it and **turn it into value**.

## Global data volume

*[Rivery, 2026]*

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

**2025**: **181 ZB** — **2028 (projection)**: **394 ZB**.

# Big Data

## What is Big Data?

**Big Data** refers to **massive data**, varied and produced at high speed, that conventional systems can no longer process efficiently.
It is characterized by the "3Vs":

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

The main challenges are to:

- collect and organize this massive flow & limit inconsistencies and redundancies,
- exploit the data to create **added value** (decision-making, prediction, automation, **AI**).

## Big Data market

*[Grand View Research, 2024]*

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

# Why use databases (DB)?

To handle the information explosion, organizations set up *data pipelines*: collection → transformation → storage → analysis. In this context, **Database Management Systems (DBMS)** play a central role. They provide:

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

## Usage examples

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

## DBMS market *[Gartner, 2024]*

**2023**: around **$100 bn** of worldwide DBMS spending,
growing by double digits every year, driven by the cloud (DBaaS).

# DB usage example: iCloud (Apple)

## What is iCloud?

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

Technically, iCloud relies on:

- **data centers** spread around the world to ensure availability,
- **relational databases** (contacts, calendars) and **NoSQL databases** (photos, videos),
- **enhanced security**: data encryption and Apple ID authentication.

## Conceptual takeaway

**iCloud** shows that **databases** are indispensable 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 varied 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 the major players (**Google**, **Apple**, **Meta**, **Amazon**, **Microsoft**).

## Link with AI (artificial intelligence)

AI models such as **ChatGPT** and **DeepSeek** are trained on huge 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**.

# Exercise 1: the 3 Vs of Big Data

**Question:** match each "V" (1 to 3) with its definition (a to c).

| **Characteristic** | **Definition** |
|---|---|
| 1. Volume | a) many formats: text, images, videos, sensors |
| 2. Velocity | b) huge amounts of data, produced continuously |
| 3. Variety | c) real-time production and processing |

Review: Big Data

# Solution of exercise 1

**Answer:** 1-b, 2-c, 3-a.

- **Volume**: the **amount** of data (149 ZB in 2024).
- **Velocity**: the **speed** of production and processing (real time).
- **Variety**: the **diversity of formats** (text, images, videos, sensors).

# Exercise 2: true or false?

**Question:** is each statement true or false?

**a)** A DBMS provides structured and durable storage of data.

**b)** iCloud uses only relational databases.

**c)** Relational databases guarantee integrity through constraints and ACID transactions.

**d)** NoSQL databases favour flexibility and scalability.

Review: iCloud and Diversification of databases

# Solution of exercise 2

a) **True**: together with security, consistency and fast, controlled access.

b) **False**: relational databases for contacts and calendars, NoSQL databases for photos and videos.

c) **True**: this is their strength (example: bank transactions).

d) **True**: they suit massive volumes (social networks, Big Data).

# Recall: the introduction

At the start of each section, three questions on the previous section.
Answer **from memory**, then click "Review" to check.

1. What do the "3 Vs" of Big Data refer to?
   Review: Big Data
2. Name three services that a DBMS provides to organisations.
   Review: Why use databases?
3. In an AI project, what is stored in a relational database, and what is stored elsewhere?
   Review: Diversification of databases

# Basic database concepts (1/9)

**Data**

A raw, encoded fact, with no particular meaning when taken in isolation.

*Examples:* 37; “Skikda”; “2026-09-28”.

**Information**

Data **interpreted by a user (or an application)** in a given **context**, so that it becomes meaningful with respect to a need.

*Examples:* 37 = a person’s age; “Skikda” = city of birth.

**Metadata**

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

*Example:* a `Client` table with a `Name` column (text type, length 50).
*NB: the DBMS manages storage, integrity and access; it does not “understand” the meaning of the data.*

*[Silberschatz, Korth & Sudarshan (2020)]*

# Basic database concepts (2/9)

## Database (DB)

A structured and consistent set of information, organized to be:

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

A relational database is built 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** = Hardware (server)

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

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

# Basic database concepts (3/9)

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

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

## Main functions

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

## Major strengths

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

# Basic database concepts (4/9)

## Schema

**Structural and permanent** representation of the database. It describes the **logical objects** (tables, columns, types, constraints, views) and their relationships. The schema serves as the **blueprint** from which the data is organized.

*Example:* a table `Student(StudentID, Name, BirthDate, Major)` is part of the university's logical schema.

## Instance

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

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

*[Silberschatz, Korth & Sudarshan (2020)]*

# Basic database concepts (5/9)

**Entity**

A real or conceptual object that is modeled in a database.

*Example:* a student.

**Attribute**

A property characterizing an entity or a relationship.

*Examples:* `Name`, `BirthDate`.

**Identifier (primary key)**

An attribute or combination of attributes that uniquely identifies each entity.

**Relationship**

A conceptual link between entities in a conceptual data model.

*[Silberschatz, Korth & Sudarshan (2020)]*

# Basic database concepts (6/9)

## Relation

A logical table made up of a set of columns (attributes) and rows (tuples). Each row represents an occurrence of an entity or of a relationship.

## Predicate

A logical statement that every tuple makes true when it belongs to the relation.

## Integrity constraints

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

*From conceptual to relational:* each **relationship** of the E/R model becomes a **relation** (table) at implementation time. *Example:* a student enrolls in a course → table `ENROLLMENT(StudentID, CourseCode, Date)`.

*[Codd (1970); Chen (1976); Silberschatz, Korth & Sudarshan (2020)]*

# Basic database concepts (7/9)

## Relational model *[Codd, 1970]*

Foundation of modern database systems:

- Data organized into **tables**.
- **Rows** (tuples) and **columns** (attributes).
- Primary and foreign **keys** for consistency.
- **Normalization** to reduce redundancy.

## Associated formalisms

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

# Basic database concepts (8/9)

## Transaction

A set of operations executed as a **logical unit of work**. A transaction guarantees the so-called **ACID** properties:

- **Atomicity**: all operations are executed in full, or none at all.
- **Consistency**: each transaction keeps the database valid (constraints respected).
- **Isolation**: concurrent transactions do not interfere with each other.
- **Durability**: once committed, changes persist despite failures.

*Example:* a bank transfer where debiting account A and crediting account B are executed together or cancelled together.

*[Silberschatz, Korth & Sudarshan (2020)]*

# Basic database concepts (9/9)

## Index

An auxiliary data structure that speeds up access to the rows of a table. It maps the values of one or more columns to the physical location of the corresponding records. Without an index, the DBMS must scan the whole table to locate the data.

## Index types

- **B-tree**: suited to searches, sorting and ranges (the most common type).
- **Hash**: efficient for equality comparisons.
- **Bitmap**: useful for columns with few distinct values.
- **Full-text**: designed for searching words in text.

## Note

Indexes improve read performance, but maintaining them can slow down update or insert operations. They are part of the physical structures managed by the DBMS.

*[Silberschatz, Korth & Sudarshan (2020)]*

# Structured Query Language (SQL)

**SQL** *[ISO/IEC 9075]*

A **standardized** language to **define**, **manipulate**, **control** and **secure** the data of a relational DB.

**Sub-languages:**

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

# Historical databases

## Historical landmarks

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

## Key takeaway

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

# Exercise 3: data, information or metadata?

**Question:** for each item, say whether it is a piece of **data**, **information** or **metadata**.

**a)** 37

**b)** Karim is 37 years old.

**c)** The `Name` column of the `Customer` table is of type text, length 50.

**d)** "2026-09-28"

Review: Data, information, metadata

# Solution of exercise 3

a) **Data**: a raw fact, meaningless on its own.

b) **Information**: the data 37 is interpreted in a context (Karim's age).

c) **Metadata**: it describes other data (the schema of the table).

d) **Data**: an isolated date, without context.

# Exercise 4: which SQL sublanguage?

**Question:** put each command in its sublanguage: **DDL**, **DML**, **DCL** or **TCL**.

`CREATE` `SELECT` `GRANT` `COMMIT`

`ALTER` `DELETE` `REVOKE` `ROLLBACK`

Review: Structured Query Language (SQL)

# Solution of exercise 4

| **Sublanguage** | **Role** | **Commands** |
|---|---|---|
| DDL | define the structure | `CREATE`, `ALTER` |
| DML | manipulate the content | `SELECT`, `DELETE` |
| DCL | manage access rights | `GRANT`, `REVOKE` |
| TCL | control transactions | `COMMIT`, `ROLLBACK` |

# Recall: basic concepts

Answer **from memory**, then click "Review" to check.

1. What is the difference between the **schema** and the **instance** of a database?
   Review: Schema and instance

2. What does the **atomicity** of a transaction guarantee?
   Review: Transactions and ACID properties

3. What is the drawback of an index?
   Review: Index

# Limitations of relational databases

Relational systems have certain limitations in modern contexts:

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

## Consequence

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

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

# NoSQL: another approach to data

**Definition** *[Sadalage & Fowler, 2013]*

The term **NoSQL** stands for *"Not Only SQL"*. It refers to a family of data management systems that offer:

- **schema flexibility** — structures can evolve freely;
- **horizontal scalability** — data can be spread across several servers;
- high performance for certain specific uses (real time, massive data, etc.).

**Some families and use cases:**

- **Key–Value** — (Redis, DynamoDB): ultra-fast storage, cache, user sessions.
- **Documents** — (MongoDB, CouchDB): semi-structured data (JSON/BSON), web apps.
- **Graphs** — (Neo4j, OrientDB): representation of complex networks (relationships, social networks).

# Object-oriented databases (OODB)

## Definition

An **object-oriented database** (OODB) directly stores the **objects** of object-oriented programming, i.e. their **data** and their **methods**.

## Main characteristics

- **Encapsulation**: data and methods are grouped within the same object.
- **Inheritance**: a class can reuse and extend the properties of another.
- **Polymorphism**: the same interface can have several behaviors.
- **Object identity**: each object keeps its own identity, independent of its values.

# Object-oriented example in PostgreSQL

## Principle

PostgreSQL supports an **object–relational** approach: **composite types** describe objects, and **functions** play the role of methods.

```
CREATE TYPE car AS (
  brand         text,
  color         text,
  price         numeric,
  serial_number text
);

CREATE OR REPLACE FUNCTION tax(c car) RETURNS numeric AS $$
  SELECT c.price * 0.02;
$$ LANGUAGE sql IMMUTABLE;

SELECT tax(ROW('Toyota','red',20000,'ABC123')::car);
```

## Key idea

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

# Towards server-side programming

Beyond declarative constraints (**PRIMARY KEY**, **FOREIGN KEY**, **CHECK**), modern DBMSs such as **PostgreSQL** make it possible to centralize **business logic** directly in the server:

- **Triggers**: automatic action on an `INSERT`, `UPDATE` or `DELETE`.
- **Stored procedures**: reusable server-side business logic.
- **Views**: simplified and secure access to data.

**Coming up**

These mechanisms are covered in depth in **Chapter 2** (Advanced SQL).

# Specialized databases

## Examples of specializations

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

## Key takeaway

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

# Exercise 5: what does the query return?

**Data:** the `car` type and the `tax` function of the object-oriented example (tax = 2 % of the price).

**Question:** what does this query return?

```
SELECT tax(ROW('Renault', 'blue', 15000, 'XYZ789')::car);
```

**a)** 15000 **b)** 300.00 **c)** 0.02 **d)** an error

Choose, then run the query to check. Review: Object-oriented example in PostgreSQL

# Solution of exercise 5

**Answer: b)** 300.00.

- `ROW(...)::car` builds an **object** of type `car` (a composite type).
- The `tax` function plays the role of a **method**: it reads the `price` field of the object.
- $15\,000 \times 0.02 = 300$.

# Exercise 6: which NoSQL family?

**Question:** for each need, choose the family: **key–value**, **document**, **column** or **graph**.

1. Keep the sessions of logged-in users in memory, with ultra-fast access.
2. Find "friends of friends" in a social network.
3. Store the articles of a web application in JSON format.
4. Absorb massive writes with very high availability.

Review: Limitations of relational databases and NoSQL

# Solution of exercise 6

| **Need** | **Family** | **Example** |
|---|---|---|
| 1. Sessions, cache | key–value | Redis, DynamoDB |
| 2. Friends of friends | graph | Neo4j |
| 3. JSON articles | document | MongoDB |
| 4. Massive writes | column | Cassandra |

# Recall: advanced concepts

Answer **from memory**, then click "Review" to check.

1. What does the term **NoSQL** mean?
   Review: NoSQL

2. Name two limitations of relational databases.
   Review: Limitations of relational databases

3. In PostgreSQL, what plays the role of an object's methods?
   Review: Object-oriented example in PostgreSQL

# 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) committees, defines a **three-level architecture** for structuring a **DBMS**.

## The three levels

- **Internal (physical) level**: describes the actual storage (*files*, *blocks*, *indexes*).
  $\Rightarrow$ *Goal:* optimize performance and the use of hardware resources.
- **Conceptual (global logical) level**: provides an abstract, unified view independent of storage (*tables*, *relationships*, *constraints*).
  $\Rightarrow$ *Goal:* guarantee consistency and logical independence.
- **External level (user views)**: offers representations tailored to the needs of users or applications.
  $\Rightarrow$ *Goal:* simplify access and strengthen security.

# Client–server architecture

## General principle

*[Silberschatz, Korth & Sudarshan, 2020]*

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

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

## How it works

*[Silberschatz, Korth & Sudarshan, 2020]*

1. Opening a network connection (TCP/IP socket).
2. Transmission of queries and responses, usually in SQL.
3. Closing the socket at the end of the session.

## Advantages

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

# Distributed architectures and databases

## Principle

When a single server is no longer enough, data is spread across several **interconnected nodes**, but managed as a **single DBMS**. Goals: availability, fault tolerance, performance.

## Key mechanisms

- **Fragmentation**: each site manages a distinct part of the data (e.g. by region).
- **Replication**: duplicating fragments for reliability and fast access.
- **Transparency**: the user queries the database without knowing where the data is located.
- **Consistency**: synchronization of copies (strong, eventual or hybrid).

## Examples

**Google Spanner**, **Amazon DynamoDB**: automatic replication and transparency.

# On-premises 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, as well as accessibility limited to the local infrastructure.

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

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

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

$\Rightarrow$ **Cloud computing** has become the preferred solution for most companies today.

# Open-source databases

**Definition** *[DataBird, 2025]*

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

It stands out for its **flexibility**, its **transparency** and the **absence of license fees**, with **continuous improvement** driven by an active community.

Its main limitations are **mostly community-based technical support**, **sometimes complex administration** and **documentation that varies from project to project**.

**Representative examples**

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

# Exercise 7: which ANSI/SPARC level?

**Question:** to which level (**internal**, **conceptual** or **external**) does each item belong?

**a)** The files, blocks and indexes on disk.

**b)** The tables, their relationships and their constraints.

**c)** A view that shows students only their name and their marks.

Review: The ANSI/SPARC model

# Solution of exercise 7

a) **Internal** (physical): the actual storage; goal: performance.

b) **Conceptual** (global logical): a unified view, independent of storage; goal: consistency.

c) **External**: a view suited to one user; goal: simpler access and better security.

# Exercise 8: which distribution mechanism?

**Question:** match each situation with a mechanism: **fragmentation**, **replication**, **transparency** or **consistency**.

1. Algiers customers are on the Algiers server, Oran customers on the Oran server.
2. The same product catalogue is copied to three servers.
3. The user queries the database without knowing where the data are.
4. After an update, the copies are synchronised so that they stay identical.

Review: Distributed architectures and databases

# Solution of exercise 8

1. **Fragmentation**: each site manages a distinct part of the data (here, by region).
2. **Replication**: copies for reliability and fast access.
3. **Transparency**: the location of the data is hidden from the user.
4. **Consistency**: the synchronisation of the copies (strong, eventual or hybrid).

# Recall: architectures

Answer **from memory**, then click "Review" to check.

1. What are the three levels of the ANSI/SPARC model?
   Review: The ANSI/SPARC model
2. How does a client communicate with a PostgreSQL server?
   Review: Client–server architecture
3. Name two advantages of a cloud-hosted database (DBaaS).
   Review: On-premises versus cloud deployment

# MVCC: Multiversion Concurrency Control

In addition to **transactions**, modern DBMSs use **MVCC (Multiversion Concurrency Control)** to manage concurrency. It consists in keeping **several versions of the same data item** so as to allow parallel execution that is both **fast** and **consistent**.

**Principle:**

- Each transaction accesses a **snapshot** of the database taken when it starts.
- When a data item is modified, a **new version** is created without deleting the old one.
- Reads see a **stable and consistent** database, while writes produce new versions.

**Consequences:**

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

# Concurrency: problems and solutions

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

- **Dirty Read**: reading data that has been modified but not yet committed.
- **Lost Update**: an update is overwritten by another transaction.
- **Non-repeatable Read**: a value changes between two successive reads.
- **Phantom Read**: rows appear or disappear between two identical queries.
- **Deadlock**: circular blocking where two transactions wait for each other.

**Control mechanisms:**

- **2PL (Two-Phase Locking)**: locks are acquired then released according to a strict order $\Rightarrow$ guarantees **serializability**.
- **MVCC (Multiversion Concurrency Control)**: each transaction accesses a **consistent snapshot** $\Rightarrow$ allows parallelism without blocking reads.
- **Deadlock detection**: the DBMS identifies cycles and aborts a transaction via `ROLLBACK`.

# Crash recovery

After concurrency control, another major challenge is **crash recovery**. Failures (power outage, software crash, disk failure) are inevitable. The DBMS must be able to:

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

**Recovery mechanisms:**

- **WAL (Write-Ahead Logging)**: log operations before they are permanently written.
- **Checkpoints**: record intermediate restart points.
- **Redo / Undo**: replay committed transactions and cancel incomplete ones.
- **Regular backups**: limit data loss in the event of a disaster.
- **DRP (Disaster Recovery Plan)**: guarantee continuity after a major failure.

# Transactional reliability and security

**Objective**

Guarantee that operations on the database are **reliable**, **consistent** and **recoverable**, even in the event of failures or concurrency.

| Mechanism | Main role | Main benefit |
|---|---|---|
| **2PL** | Lock control to avoid conflicts between transactions. | **Serializability** |
| **MVCC** | Concurrent access through several versions of the data. | **High parallelism** |
| **WAL** | Logging of operations beforehand. | **Reliable recovery** |
| **Checkpoints** | Periodic saving of a consistent state. | **Fast restart** |
| **DRP** | Organizing recovery after a major disaster. | **Service continuity** |

*These mechanisms are the pillars of DBMS reliability.*

# Exercise 9: which anomaly?

**Question:** name the concurrency anomaly in each situation.

1. T1 reads a balance that T2 has changed but not yet committed; then T2 rolls back.
2. T1 and T2 both read 100, each adds 10, and both write 110.
3. T1 reads 100; T2 changes the value to 150 and commits; T1 reads 150 again.
4. T1 counts 5 rows; T2 inserts a row and commits; T1 counts 6 rows again.
5. T1 waits for a lock held by T2, and T2 waits for a lock held by T1.

Review: Concurrency: problems and solutions

# Solution of exercise 9

1. **Dirty Read**: T1 read a value that was never committed.
2. **Lost Update**: the addition of one of the two transactions is lost (110 instead of 120).
3. **Non-repeatable Read**: the same row changes between two reads.
4. **Phantom Read**: a "phantom" row appears between two identical queries.
5. **Deadlock**: a circular wait; the DBMS rolls back one of the two (`ROLLBACK`).

# Exercise 10: which mechanism?

**Question:** for each need, choose the mechanism: **2PL**, **MVCC**, **WAL**, **checkpoint** or **DRP**.

1. After a power cut, redo the committed transactions.
2. Restart quickly, from a recent consistent state.
3. Reads must not block writes.
4. Guarantee serializability with locks.
5. Resume activity after a fire in the data centre.

Review: Transactional reliability and security

# Solution of exercise 10

1. **WAL**: operations are logged before being written for good, then redone (*redo*).
2. **Checkpoint**: a saved restart point; recovery starts from it.
3. **MVCC**: each transaction reads its own version (*snapshot*).
4. **2PL**: locks are acquired, then released, in a strict order.
5. **DRP**: the disaster recovery plan after a major incident.

# Conclusion (chapter summary)

- **Data** is the **heart of the modern digital world** (it powers services, AI and decision-making).
- **Models** have diversified from **relational** to **NoSQL**, by way of **object–relational**.
- **Architectures** have 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)** $\rightarrow$ **394 ZB (2028, projection)**.
- Big Data market estimated at **$862.3 billion by 2030**; DBMS market around $100 bn (2023).

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

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