

1 · WHY DBMS > FILE SYSTEM

DB = large integrated collection of data modelling a real-world enterprise (entities + relationships). DBMS = software to store & manage it.

6 drawbacks of file systems

1. Data redundancy & inconsistency — many formats, duplicated info

2. Difficulty accessing data — new program for every new task

3. Data isolation — multiple files & formats

4. Integrity problems — constraints buried in code, hard to change

5. Atomicity of updates — crash → partial update (transfer half done)

6. Concurrent access anomalies — 2 users read 100, each withdraw 50

7. Security problems — can't give access to some data only

Why use a DBMS (7)

Reduce app dev time · Data independence · Efficient data access · Concurrent access · Crash recovery · Uniform data administration · Data integrity · Security

2 · LEVELS OF ABSTRACTION (3-LEVEL ARCHITECTURE)

Level	Describes
Physical	HOW a record is stored (files, indexes, bytes)
Logical	WHAT data is stored + relationships (the schema)
View	Part of DB one app/user sees; hides data (e.g. salary) for security

Physical data independence = change physical schema WITHOUT changing logical schema. (Apps depend on logical schema.)

Schema vs Instance — like type vs variable.

Schema = logical structure/design (rarely changes). Instance = actual content at a point in time (changes constantly). Insert 1 row → instance changes, schema does NOT.

3 · DATA MODELS & LANGUAGES

Data model = high-level constructs hiding low-level storage. Models: Relational (dominant) · ER (for design) · Object-based · Semistructured (XML) · NoSQL (doc/key-value/column-family/graph) · older: network, hierarchical.

DDL	Define schema, types, integrity constraints, indexes, authorization, storage. Output → data dictionary (metadata = data about data)
DML	= query language. Query/insert/delete/modify. Pure: relational algebra, tuple/domain relational calculus. Commercial: SQL

Database Engine (3 parts)

Storage manager (interface to OS files; storage access, file org, indexing/hashing) · Query processor (1 parse+translate → 2 optimize → 3 evaluate) · Transaction manager (atomicity + recovery; concurrency-control manager)

Transaction = collection of operations performing a single logical function. ACID = Atomicity, Consistency, Isolation, Durability.

Architectures: centralized · client-server · parallel · distributed.

NoSQL: no fixed schema, denormalized to avoid joins, scales horizontally (MongoDB/Cassandra/Neo4j/Redis). NewSQL: NoSQL scale + ACID + SQL (Spanner, CockroachDB).

4 · RELATIONAL MODEL

Codd, 1970. Relation=table, tuple=row, attribute=column, domain=allowed values, arity=#attributes, cardinality=#tuples.

Attribute values must be atomic (indivisible) → 1NF

null ∈ every domain = unknown / not exist

Relations are sets → tuple order is irrelevant, no duplicate tuples

Schema: instructor = (ID, name, dept_name, salary)

5 · KEYS (EXAM FAVOURITE)

Key	Definition
Superkey K	Set of ≥1 attributes that uniquely identifies a tuple. If t1≠t2 then t1.K≠t2.K. Any superset of a superkey is a superkey.
Candidate key	Minimal superkey — no proper subset is a superkey
Primary key	The candidate key chosen by the designer. Should never/rarely change. Written first & underlined
Foreign key	Attribute(s) in r1 that must appear as PK of r2. r1 = referencing, r2 = referenced

Student(student_id, national_id, email, full_name)
Superkeys : {student_id}, {national_id}, {email}, {student_id,name}, {student_id,email}, ... (any set containing a candidate key)
Candidate : {student_id}, {national_id}, {email}
Primary : student_id (email may change; nat_id = privacy)

Trap: {student_id, national_id} is a superkey but NOT candidate — you can drop one attribute and it still identifies uniquely (not minimal).

FK constraint prevents (a) breaking links between tables and (b) inserting invalid values. Insert child row with non-existent parent → error Cannot add or update a child row: a foreign key constraint fails.

Composite PK e.g. Enrollment(student_id, course_id, semester, year) — student_id alone can't be PK because one student enrolls in many courses.

6 · RELATIONAL ALGEBRA (6 BASIC OPERATORS)

SQL says WHAT; algebra is HOW the DB thinks. Every SQL query is translated to relational algebra, then the optimizer picks the cheapest equivalent plan.

Closure property: relation in → relation out ⇒ operators compose into a tree.

Op	Sym	Meaning / Notes
Select	σ	rows. σ _r (r). p uses = ≠ > ≥ < ≤ and ∧ ∨ ¬. Can compare 2 attributes: σ _{dept_name=building} (department)
Project	π	columns. π _{A1,...,Ak} (r). Removes duplicate rows (result is a set!)
Union	∪	needs same arity + compatible domains
Difference	−	in r not in s. Same requirements
Cartesian	×	every pair. Common attr renamed instructor.ID
Rename	ρ	ρ _x (E) or ρ _x (A1,...,An)(E)

Derived: n (intersection) — note r ∩ s = r − (r − s) ∩ (natural join) · ← (assignment to temp relation).

π name (σ dept_name='Physics' (instructor))
σ dept_name='Physics' ∧ salary>90000 (instructor)
π course_id(σ sem='Fall'∧year=2017(section))
∪ / n / − π course_id(σ sem='Spring'∧year=2018(section))
Join = σ instructor.ID = teaches.ID (instructor × teaches)
Physics − σ dept_name='Physics' (instructor) -- assignment
p AvailableStock + StockQuantity (Inventory) -- rename

Equivalence / optimization

σ_{AB}(r) ≡ σ_A(σ_B(r)) → EQUIVALENT (same result on any instance).

π_{name}(σ_{gpa>3}(S)) vs σ_{gpa>3}(π_{name}(S)) → NOT equivalent: 2nd projects gpa away first, so σ can't be applied (invalid).
Optimizer rule: push σ down — filter first, then join (less data) = faster.

SQL + algebra: SELECT→π · WHERE→σ · FROM (multi-table)→× · JOIN...ON→⋈ · UNION/INTERSECT/EXCEPT→∪/∩/−

7 · SQL DDL — CREATE TABLE

CREATE DATABASE db; USE db; SELECT DATABASE(); SHOW DATABASES; SHOW TABLES; DESCRIBE t;

CREATE TABLE instructor (
ID CHAR(5),
name VARCHAR(20) NOT NULL,
dept_name VARCHAR(20),
salary NUMERIC(8,2),
PRIMARY KEY (ID),
FOREIGN KEY (dept_name) REFERENCES department(dept_name)
);
CREATE TABLE section (
course_id VARCHAR(8), sec_id VARCHAR(8),
semester VARCHAR(6), year NUMERIC(4,0),
PRIMARY KEY (course_id,sec_id,semester,year)); -- composite

Constraints: PRIMARY KEY · FOREIGN KEY...REFERENCES · NOT NULL · UNIQUE · CHECK · DEFAULT · AUTO INCREMENT

ALTER TABLE t ADD PRIMARY KEY (col);
ALTER TABLE t ADD CONSTRAINT fk_n FOREIGN KEY (c) REFERENCES parent(c);
DROP TABLE t; DELETE FROM t; INSERT INTO t VALUES (-);

DELETE vs DROP — DELETE removes data only (structure stays, recoverable, slow on big tables). DROP removes data + structure (faster to drop&recreate).

8 · SELECT — FROM — WHERE

SELECT [DISTINCT ALL] A1, A2, ... FROM r1, r2, ... (product) WHERE p ORDER BY col [ASC DESC], col2; -- π (projection) -- × (Cartesian) -- σ (selection)	• SQL keeps duplicates by default (multiset/bag) → DISTINCT removes them; ALL keeps (default)	• SQL names are case-insensitive: name ≡ NAME ≡ Name. But string data & LIKE patterns ARE case-sensitive	• SELECT * = all attributes · SELECT '437' AS F00 = literal column	• Arithmetic in select: SELECT ID,name,salary/12 AS monthly_salary	• Self-join / rename: FROM instructor AS T, instructor AS S (AS optional)	• Result of any SQL query is a relation
--	---	--	--	--	---	---

-- Cartesian product (no condition) vs Join (with condition)
SELECT name, department.dept_name FROM instructor, department;
SELECT name, department.dept_name FROM instructor, department
WHERE instructor.dept_name = department.dept_name;

9 · WHERE PREDICATES

Predicate	Example
Comparison	= <> != < > <= >=
BETWEEN	salary BETWEEN 60000 AND 80000 (Inclusive both ends)
IN	country IN ('USA','UK')
LIKE	%=any substring (0,1,many) · _=exactly 1 char
IS [NOT] NULL	WHERE ship_date IS NULL (never = NULL)
AND OR NOT	cat='E' AND (price<500 OR stock>100)
EXISTS	EXISTS (SELECT 1 FROM e WHERE e.d_id=d.id)
ANY/SOME	true if at least one subquery value satisfies
ALL	true if every subquery value satisfies
REGEXP	full regex, not standard SQL (MySQL/PG). SIMILAR TO = PG only

'Intro%' → starts with Intro
'%Comp%' → contains Comp
'_ _ _' → exactly 3 characters
' _ _ %' → at least 3 characters
'The%Adventure' → starts The AND ends Adventure
' _ of' → exactly 3 chars, then " of"
LIKE '100 %' ESCAPE '\ ' -- match a literal % sign

Strings: || concat · UPPER/LOWER · LENGTH · SUBSTRING. Tuple compare: (a,b) = (x,y)

10 · JOINS

Type	Returns
INNER	only matching rows from both tables
LEFT [OUTER]	ALL left rows + matches; unmatched right cols = NULL
RIGHT [OUTER]	ALL right rows + matches; unmatched left cols = NULL
FULL [OUTER]	all rows from both. MySQL has NO FULL JOIN → simulate: LEFT UNION RIGHT
NATURAL	auto-matches on all common column names, keeps one copy of each common column (no ON clause)
CROSS / FROM a,b	Cartesian product (every pair)

SELECT * FROM a INNER JOIN b ON a.id = b.a_id;
SELECT * FROM a LEFT JOIN b ON a.id = b.a_id;
SELECT * FROM a NATURAL JOIN b;
-- FULL OUTER JOIN in MySQL:
SELECT * FROM a LEFT JOIN b ON a.id=b.a_id
UNION
SELECT * FROM a RIGHT JOIN b ON a.id=b.a_id;

Join condition = which tuples match. Join type = how NON-matching tuples are treated.

-- employees who earn more than their manager (self-join)
SELECT e1.first_name, e1.salary, e2.first_name AS mgr
FROM employees e1 JOIN employees e2
ON e1.manager_id = e2.employee_id
WHERE e1.salary > e2.salary;

11 · SCHEMA DESIGN RULES

• snake_case, not CamelCase (portable, case-insensitive, matches SQL)

• Tables = plural nouns (customers, orders) · columns = singular

• PK = table_name_id · FK uses the same name as the PK it references

• Dates * _date · Booleans is_ / has_

• Decompose composite attributes when parts are queried/sorted separately

• Domain / CHECK constraints enforce business rules in the DB

CREATE TABLE students (
student_id INT PRIMARY KEY,
student_age INT CHECK (student_age BETWEEN 16 AND 150),
current_grade ENUM('A','B','C','D','F'),
gpa DECIMAL(3,2) CHECK (gpa >= 0.0 AND gpa <= 4.0),
email VARCHAR(100) CHECK (email LIKE '%@%.%'),
enrollment_date DATE NOT NULL, graduation_date DATE,
CONSTRAINT chk_grad CHECK (graduation_date IS NULL OR graduation_date > enrollment_date));

Bad-design symptoms: repeating groups (product1,product2,...)
• text blob instead of columns · redundancy → update anomalies · missing index → full scan · no transaction → money lost on crash.

12 · SET OPERATIONS

UNION	combine, removes duplicates (UNION ALL keeps them)
INTERSECT	rows in both
EXCEPT (MINUS)	rows in 1st but not in 2nd

(SELECT course_id FROM section WHERE sem='Fall')
INTERSECT
(SELECT course_id FROM section WHERE sem='Spring');

Both SELECTs need same number of columns + compatible types.
MySQL 8.0.31+ has INTERSECT/EXCEPT; older MySQL → simulate with IN / NOT IN / joins.

13 · NULL & THREE-VALUED LOGIC ★

- null = **unknown** or **does not exist** (NOT zero, NOT empty string)
- Any arithmetic with null → null:** 5 + NULL = NULL, manager_id+100 = NULL
- Any comparison with null → unknown:** 5<NULL, NULL=NULL, NULL<NULL all **unknown**
- WHERE keeps a row **only if the predicate is TRUE**. unknown → row dropped
- Test with IS NULL / IS NOT NULL only

OR	T	F	U		AND	T	F	U
T	T	T	T		T	T	F	U
F	T	F	U		F	F	F	F
U	T	U	U		U	U	F	U

NOT unknown = unknown. "P IS UNKNOWN" is TRUE when P evaluates to unknown.

Classic trap: WHERE manager_id <> 1 — employees with **manager_id = NULL are MISSING** from the result, because NULL<>1 → unknown → not TRUE. Fix: WHERE manager_id <> 1 OR manager_id IS NULL.

14 · AGGREGATES · GROUP BY · HAVING

AVG · MIN · MAX · SUM · COUNT — operate on a **multiset** (bag: order irrelevant, duplicates allowed) of column values → 1 value.

NULL rule: ALL aggregates EXCEPT COUNT(*) IGNORE NULLS.
COUNT(*) = every row · COUNT(col) = non-null values only · COUNT(DISTINCT col) = unique non-null.
If a column has ONLY nulls: **COUNT → 0**, all others (SUM/AVG/MIN/MAX) → **NULL**.

SELECT dept, COUNT(*) AS n, AVG(salary) AS avg_s FROM employees
WHERE hire_date >= '2019-01-01' -- filters ROWS (before grouping)
GROUP BY dept
HAVING COUNT(*) > 1 -- filters GROUPS (after grouping)
ORDER BY avg_s DESC;

	WHERE	HAVING
Applied	before GROUP BY	after GROUP BY
Filters	individual rows	groups
Aggregates?	NO	YES

Rule: every attribute in SELECT that is **outside** an aggregate function **must** appear in GROUP BY.

Logical execution order: **FROM → WHERE → GROUP BY → HAVING → SELECT → DISTINCT → ORDER BY → LIMIT**

15 · NESTED SUBQUERIES

Can appear in **WHERE, FROM, SELECT**.

```
-- WHERE: set membership
... WHERE course_id IN (SELECT course_id FROM ...);
-- WHERE course_id NOT IN (SELECT ...);
-- set comparison
... WHERE salary > SOME/ANY (SELECT salary FROM e WHERE dept='HR');
-- > at least ONE (i.e. > MIN)
... WHERE salary > ALL (SELECT salary FROM e WHERE dept='HR');
-- > EVERY one (i.e. > MAX)
-- EXISTS / correlated subquery
SELECT * FROM employees e
WHERE EXISTS (SELECT 1 FROM employees r
               WHERE r.manager_id = e.employee_id);
-- correlated: inner query uses value from outer query (e)
-- FROM: derived table (MUST have an alias)
SELECT dept, avg_s FROM
  (SELECT dept, AVG(salary) AS avg_s FROM employees
   GROUP BY dept) AS d
WHERE avg_s > 70000;
-- WITH: temporary relation, visible only to this query
WITH dept_total(dept,total) AS
  (SELECT dept, SUM(salary) FROM employees GROUP BY dept)
SELECT * FROM dept_total WHERE total > 150000;
-- SELECT: scalar Subquery (must return exactly 1 row, 1 col)
SELECT d.dept name,
  (SELECT COUNT(*) FROM instructor i
   WHERE i.dept_name=d.dept_name) AS n
FROM department d;
```

16 · MODIFICATION · CASE · COALESCE

```
DELETE FROM instructor; -- all rows
DELETE FROM instructor WHERE dept_name='Finance';
DELETE FROM instructor WHERE dept_name IN
  (SELECT dept_name FROM department WHERE
   building='Watson');

INSERT INTO course VALUES ('CS-437','DB Systems','Comp. Sci.',4);
INSERT INTO course (course_id,title,dept_name,credits)
VALUES (-);
INSERT INTO student VALUES ('3003','Green','Finance',NULL);
INSERT INTO student SELECT ID,name,dept_name,0 FROM
instructor;
-- SELECT is evaluated FULLY before any row is inserted

UPDATE instructor SET salary = salary*1.03 WHERE
salary>100000;
UPDATE instructor SET salary = salary*1.05 WHERE
salary<=100000;
-- ORDER MATTERS above! (do 3% first, else some get
both raises)
-- Better — one statement, no order problem:
UPDATE instructor SET salary = CASE
  WHEN salary > 100000 THEN salary*1.03
  ELSE salary*1.05 END;
```

COALESCE(e1,e2,...) returns the **first non-NULL** value (all null → NULL).
SELECT COALESCE(phone,'No phone') AS contact FROM employees;

17 · VIEWS

A **virtual relation** — hides data (e.g. salary) from users & simplifies queries. **Not stored:** the definition (an expression) is saved and substituted into queries.

```
CREATE VIEW faculty AS
  SELECT ID,name,dept_name FROM instructor;
CREATE VIEW dept_total(dept_name,total_salary) AS
  SELECT dept_name, SUM(salary) FROM instructor GROUP
  BY dept_name;
SELECT name FROM faculty WHERE dept_name='Biology';
DROP VIEW [IF EXISTS] faculty; -- NOT "DELETE"
(DELETE = rows only)
```

A view may be defined **using another view**; v1 *depends on* v2. A view depending on itself is **recursive**. Drop dependent views first.

Updatable view — most SQL allows INSERT/UPDATE only if:
FROM has **exactly one** relation · SELECT lists only plain attribute names (no expressions / aggregates / DISTINCT) · **no GROUP BY / HAVING**. Attributes not listed become NULL.
Some updates **cannot be translated uniquely** (e.g. inserting into a join view — which department is "Taylor"?).

	View	Materialized View
Stored?	No (query re-run)	Yes — physical copy of the result
Speed	slower reads	fast reads
Freshness	always current	goes stale ; must be maintained

18 · DATA TYPES

INT / SMALLINT / BIGINT	whole numbers (INT ≈ 2 billion)
DECIMAL(p,s) / NUMERIC(p,d)	exact — use for MONEY. numeric(3,1): 44.5 ok, 444.5 x
FLOAT / REAL / DOUBLE	approximate — never for money (19.95 → 19.950000000000000003)
CHAR(n)	fixed length (ISBN CHAR(13))
VARCHAR(n)	variable length
TEXT / ENUM	long text · ENUM('A','B','C') list of allowed values
DATE · TIME · TIMESTAMP · INTERVAL	DATE '2005-7-27' · TIME '09:00:30' · TIMESTAMP=date+time · date-date = interval
BLOB / CLOB	binary / character large object (query returns a pointer , not the object)
BOOLEAN · JSON · XML · UUID	

Common mistakes: VARCHAR(255) for everything (waste) · FLOAT for money (precision) · TEXT for short strings (slow) · forgetting NOT NULL.

19 · INDEXES

```
CREATE INDEX idx_email ON customers(email);
CREATE INDEX studentID_index ON student(ID);
```

	No index	With B-tree index
Method	full table scan	direct lookup (3–4 levels)
Complexity	O(n) linear	O(log n) logarithmic
1M rows	≈ 2.3 s	≈ 0.003 s

B-tree = self-balancing tree, keeps data sorted, logarithmic search. Index stores sorted values + row pointers, **stored separately** from table data.

Create an index on: columns used in WHERE · columns used to JOIN · large tables queried often · **high-cardinality** columns (many distinct values).

Trade-off: faster reads, **slower writes** (INSERT/UPDATE/DELETE must update the index) + extra storage. PK gets an index automatically.

20 · AUTHORIZATION

```
GRANT <privileges> ON <relation/view> TO
<users>[roles][PUBLIC];
REVOKE <privileges> ON <relation/view> FROM <users>;

GRANT SELECT ON instructor TO U1,U2,U3;
GRANT SELECT,INSERT,UPDATE,DELETE ON employees TO
hr_team;
GRANT UPDATE (email,phone) ON employees TO intern_role;
-- col columns
REVOKE ALL PRIVILEGES ON DATABASE company_db FROM
john_doe;

CREATE ROLE instructor; GRANT instructor TO Amit;
GRANT SELECT ON takes TO instructor;
CREATE ROLE dean; GRANT instructor TO dean; GRANT dean TO Satoshi;
```

Privileges: **SELECT · INSERT · UPDATE · DELETE · EXECUTE · ALL PRIVILEGES**. Roles can be granted to users **and to other roles** (inheritance chain).

- Grantor must already hold the privilege (or be DBA)
- Granting on a **view** does **NOT** grant anything on the underlying tables — and the user does **not** need base-table rights. The view is a **security layer**
- REVOKE also revokes all privileges that **depend** on the revoked one; user may keep it if granted twice by different grantors
- Principle of **least privilege**

21 · ER MODEL (SILBERSCHATZ NOTATION)

Symbol	Meaning
□ rectangle	entity set
□= double rectangle	weak entity set
◇ diamond	relationship set
◇= double diamond	identifying relationship (to weak entity)
ellipse	attribute · <u>underline</u> = PK · <u>dashed underline</u> = discriminator / partial key
— single line	partial participation (may not take part)
= double line	total participation (every entity MUST take part)
→ arrow	points to the "one" side

Entity or Attribute? → Entity if it has its own sub-attributes, can occur many times per parent, or has its own identity (Customer, Address, Review, Category). **Attribute** if it is a single atomic value (loyalty_points, star_rating).

Weak entity = cannot exist without its owner & has no PK of its own; identified by **owner PK + discriminator** (e.g. Chapter(book_id, chapter_no), Vaccination(pet_id, dose_no)).

Total: every employee **MUST** belong to a dept (= on Employee side). **Partial:** some depts may have no employees.

22 · ER → RELATIONAL SCHEMA ★★

ER component	Mapping
Entity	Table; attributes → columns; PK → PRIMARY KEY
Simple attribute	one column
Composite attribute	break into separate columns (address → street, city, zip). Do NOT create a new table
Multi-valued attribute	separate table (violates 1NF otherwise). Never phone1/phone2/phone3 or CSV in one field
1 : 1	FK in either table + UNIQUE constraint on the FK
1 : N / N : 1	FK on the "many" side, referencing the "one" side
M : N	NEW junction table with FKs to both; PK = both FKs (+ relationship attributes)
Ternary	New table with FKs to all three entities + relationship attributes
Total participation	FK NOT NULL
Partial participation	FK allows NULL
Weak entity	Table with FK→owner; composite PK = (owner PK, discriminator)
Relationship attribute	Put it with the FK (1:N) or in the junction table (M:N)

-- WORKED EXAMPLE (vet clinic)
Owner(owner_id PK, name NOT NULL, street, city, zip) - composite addr
Pet(pet_id PK, pet_name NOT NULL, species, owner_id INT NOT NULL FK-Owner(owner_id)) -- 1:N + TOTAL
Vet(vet_id PK, vet_name NOT NULL, specialty)
Treatment(-- M:N Pet-Vet, repeatable, with attributes
 pet_id FK-Pet, vet_id FK-Vet, treat_date DATE, diagnosis VARCHAR(255),
 PRIMARY KEY (pet_id, vet_id, treat_date))
Vaccination(-- WEAK entity owned by Pet
 pet_id FK-Pet, dose_no INT, -- dose_no = discriminator
 vaccine_name VARCHAR(50) NOT NULL,
 PRIMARY KEY (pet_id, dose_no))

Marker's deduction list: making Address a separate table (–2) · forgetting NOT NULL on the total-participation FK (–1) · using an FK instead of creating the M:N junction table (–4) · weak entity given its own new id / single-column PK (–2) · leaving the repeat-date out of the PK when the same pair can meet many times (–2).

★ ลำดับการลอกกระดาษ (COPY ORDER — เขียนไม่ทันทั้งหมด ให้ลอกตามนี้)

รอบ 1 — ต้องมี (เขียนก่อน): 22 ER–Schema · 13 NULL/3-valued · 14 Aggregates+GROUP BY/HAVING · 5 Keys · 10 Joins · 15 Subqueries · 6 Relational Algebra

รอบ 2 — ถ้ามีที่เหลือ: 7 CREATE TABLE · 9 Predicates+LIKE · 21 ER symbols · 16 Modification/CASE · 17 Views · 12 Set ops

รอบ 3 — เต็มข้อกว่า: 1–4 ทฤษฎี · 18 Data types · 19 Index · 20 Authorization · 11 Naming/CHECK

เทคนิค: กล้องครอบเหลือ = จุดที่ออกสอบบ่อย/ผิดกันเยอะ ลอกให้ครบ · ได้คีย์ว ๆ ลอกแค้นใครง (keyword + ลำดับ clause) พอ · เขียนหน้าหลัง ตามลำดับนี้ จะได้ไม่ต้องพลิกไปมาจนสับสน