Databases

Curated computer science study track (not a degree, no licensure claim). Original explanations and practice only, following the reviewed course handout.

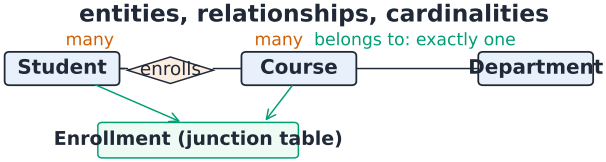
A set of true sentences about a domain

- 8.1-8.3 model and shape the data; 8.4-8.5 make it fast and safe; 8.6 governs it.
- Every query and output in this deck was executed against the documented sample schema.

Relational modelling: entities to keys

1. **Model** Student, Course, Department are entities; enrolls is many-to-many and becomes the Enrollment junction table of foreign-key pairs.
2. **Keys PRIMARY KEY** (student, course, term) rejects a duplicate triple at the engine - rules live in the schema.
3. **Join** Student to Enrollment to Course to Department follows the foreign keys: the sample data (8-6-14) yields 14 rows.

ER diagram



many-to-many becomes its own table of foreign-key pairs; PRIMARY KEY (student, course, term) enrolls once per term
referential integrity: every Enrollment.student and .course names a real row
sample data: 8 students - 6 courses - 14 enrollments

> Students enroll in Courses through the Enrollment junction table.

Relational algebra and SQL: the pipeline

1. **Pipeline Join** Course x Enrollment, select credits = 15, group by title, keep groups of 2 or more: executed result - Databases | 3.
2. **Joins** A plain join drops the 2 empty courses; LEFT JOIN keeps them at n = 0 - 6 rows; COUNT(col) skips NULLs, COUNT(*) does not.
3. **Order** FROM builds rows, WHERE filters, GROUP BY partitions, HAVING filters groups, SELECT projects, ORDER BY sorts.

Table pipeline

one query, five operators, left to right



FROM builds rows, WHERE filters rows, GROUP BY partitions, HAVING filters groups

```
SELECT c.title, COUNT(*) AS n
FROM Course c JOIN Enrollment e ON c.course_id = e.course_id
WHERE c.credits = 15 GROUP BY c.title HAVING COUNT(*) >= 2 ORDER BY n DESC;
```

executed result: Databases | 3

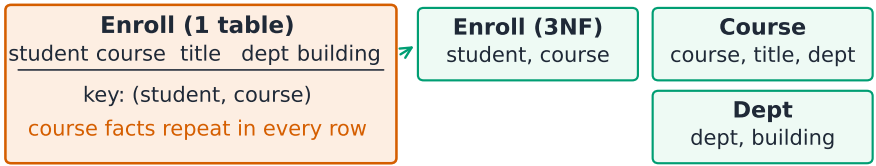
One query as a pipeline of the five algebra operators, left to right.

Normalisation: each fact once

1. **Split** Enroll(student, course, title, dept, building): course to title is part-key, title to dept to building transitive - three tables, zero anomalies.
2. **Forms** 2NF removes part-key dependencies, 3NF the transitive ones, BCNF demands every determinant be a key.
3. **Stop** A stored running total is derivable but maybe worth it: normalise to reason, denormalise on evidence.

Dependency diagrams

split by the dependencies: each fact once



FDs: course → title, dept (part of the key); dept → building (transitive)

anomalies gone: insert, update and delete each touch one row per fact

2NF drops part-key FDs; 3NF drops transitive FDs; BCNF: every determinant a key

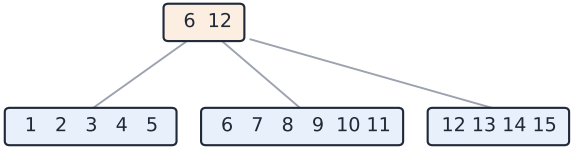
One denormalised table split by its dependencies into three.

Indexes and query plans: the plan tells the truth

1. **Tree** Order-4, 15 keys, 2 levels: any key in 2 reads; fan-out 100 covers a million keys at height 3.
2. **Plan** EXPLAIN **QUERY PLAN**: without an index, SCAN Course; with INDEX ON Course(credits), SEARCH USING INDEX - 6 rows instant, 6 million rows 4 reads versus a scan.
3. **Tax** Table + 3 trees = 4 structure writes per insert; a 5-index log at 10 million inserts daily maintains 50 million entries.

B-tree and query plan

two levels, fifteen keys



order-4 B-tree: any key in 2 reads

fan-out: 100^3 = a million keys in 3 reads

the plan tells the truth

EXPLAIN QUERY PLAN

no index:
SCAN Course

with INDEX ON Course(credits):
SEARCH USING INDEX idx (credits=?)
6 rows: both instant; 6 million: 4 reads

every index is a loan the writes repay: table + 3 trees = 4 structures per insert

Left: the **B-tree**, two levels for 15 keys. Right: the plan that uses it.

Transactions and concurrency: all or nothing

1. **Lost** T1 reads 100, T2 reads 100, T1 writes 90, T2 writes 120 - the withdrawal vanished: same race as the OS unit, now with money.
2. **Orders** R1 R2 W2 W1 ends 90 (deposit lost); R1 W1 R2 W2 ends 110 - equal to serial T1 then T2.
3. **Locks Two-phase locking** holds locks to commit: one waits, the outcome is serial-equivalent; deadlock is the price, retry the cure.

Interleaving timeline

same operations, two interleavings, two verdicts

lost update

T1 R(x) 100	T2 R(x) 100	T1 W(x) 90	T2 W(x) 120

ends 120: the -10 lost

serial-equivalent

T1 R(x) 100	T1 W(x) 90	T2 R(x) 90	T2 W(x) 110

ends 110: serial

2PL: reads and writes lock, locks held to commit - one transaction waits, the outcome equals a serial order

isolation levels name the anomalies you accept: dirty read gone at read-committed, phantoms gone only at serializable

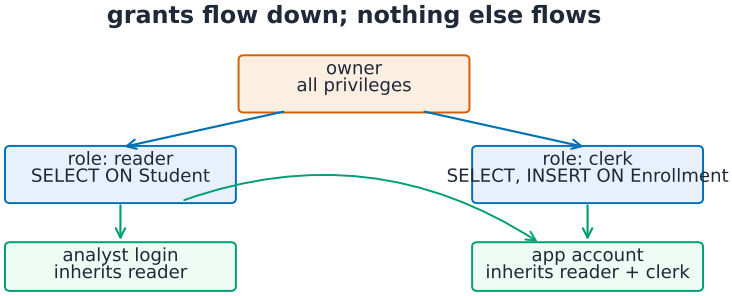
ACID: atomic - consistent - isolated - durable

> Two interleavings: one loses the update, one equals a serial order.

Security and governance: narrow the grants

1. **Inject** Pasting user text lets ' OR '1'='1 match every row - all 8 students on the sample; **parameterised statements** end the class, not the example.
2. **Grants** GRANT SELECT ON Student TO reader; SELECT, INSERT ON Enrollment TO clerk - a stolen session is bounded by the grant.
3. **Loop** Governance classifies, assigns owners, sets retention, audits access - revoke on role change, never connect the app as owner.

Access model



least privilege: the app account reads two tables and writes one - a stolen session is bounded by the grant

Grants flow from the owner to roles to users - nothing else flows.

Check yourself

- A many-to-many **relationship** becomes...? (A junction table.)
- Filtering on COUNT(*) belongs in...? (HAVING.)
- Fan-out 100, height 3 covers...? (A million keys.)
- The schedule R1 R2 W2 W1 ends at...? (90 - the deposit lost.)