

Exercise walk-through

8.3

Normalisation

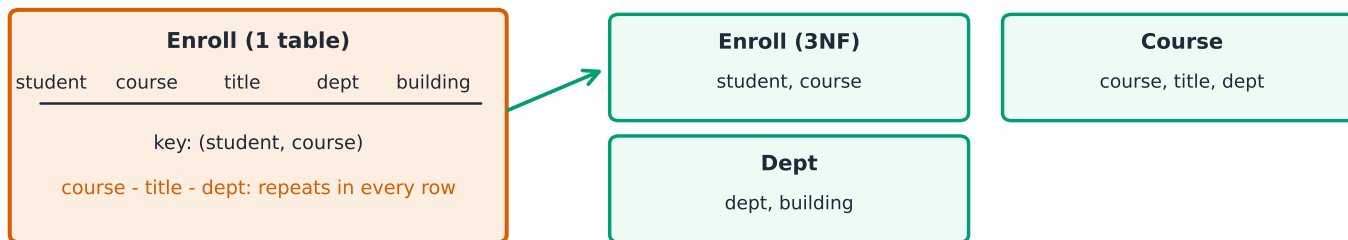
University Computer Science

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 (part-key out), title - dept - building (transitive out)

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

2NF removes part-key dependencies; 3NF removes transitive ones; BCNF: every determinant a key

One denormalised table split by its dependencies into three.

Exercises 2.1-2.3

2.1 In `Enroll(student, course, title)`, name the part-key dependency. [3]

2.2 Deleting the last enrollment for a course also deletes the course's title. Name the anomaly. [3]

2.3 A table in BCNF is automatically in which lower forms? [3]

Exercises 2.1-2.3

2.1 In `Enroll(student, course, title)`, name the part-key dependency. [3]

2.1

- `course → title` – the course determines the title from part of the key (`student, course`) ✓.

2.2 Deleting the last enrollment for a course also deletes the course's title. Name the anomaly. [3]

2.2

- The delete anomaly – the fact rides on the last row carrying it ✓.

2.3 A table in BCNF is automatically in which lower forms? [3]

2.3

- 2NF and 3NF (BCNF implies both) ✓.

Exercise 3.1

3.1 Project(proj_id, dept, dept_head, member, hours) with key (proj_id, member) and facts: each project belongs to one department; each department has one head.

List every functional dependency. Identify the part-key and the transitive dependency. Split to 3NF (give the tables), and state which anomaly each split removes. Name the form of the original table. [8]

Exercise 3.1

3.1 Project(proj_id, dept, dept_head, member, hours) with key (proj_id, member) and facts: each project belongs to one department; each department has one head.

List every functional dependency. Identify the part-key and the transitive dependency. Split to 3NF (give the tables), and state which anomaly each split removes. Name the form of the original table. [8]

3.1

- FDs: (proj_id, member) $\rightarrow$ hours; proj_id $\rightarrow$ dept; dept $\rightarrow$ dept_head ✓.
- Part-key: proj_id $\rightarrow$ dept (dept depends on half the key); transitive: proj_id $\rightarrow$ dept $\rightarrow$ dept_head ✓.
- 3NF split into three tables ✓: Project(proj_id, dept); Dept(dept, dept_head); Work(proj_id, member, hours).
- The splits remove the update anomaly (department facts stored once) and the insert anomaly (a department with no project becomes insertable) ✓.
- Original form: 1NF only (it violates 2NF) ✓.

Exercise 3.2

3.2 A report stores `order_id`, `line_no`, `product`, `unit_price`, `qty`, `order_total` with key `(order_id, line_no)`; the total is the sum of `qty x unit_price` over the order's lines.

Name the derivable column and the anomaly it invites. State the 3NF treatment and the cost it adds per read of the report. Give the condition under which keeping the total is the better engineering choice, naming the mechanism that keeps it honest. [7]

Exercise 3.2

3.2 A report stores `order_id`, `line_no`, `product`, `unit_price`, `qty`, `order_total` with key (`order_id`, `line_no`); the total is the sum of `qty` x `unit_price` over the order's lines.

Name the derivable column and the anomaly it invites. State the 3NF treatment and the cost it adds per read of the report. Give the condition under which keeping the total is the better engineering choice, naming the mechanism that keeps it honest. [7]

3.2

- Derivable: `order_total` (sum over lines) ✓.
- Anomaly: update anomaly – one line changes and every stored total must change with it; miss one and the report lies ✓.
- 3NF treatment: drop the column; compute the total per read with an aggregation – the cost is a GROUP BY over the order's lines ✓.
- Keep it when the report is read far more often than lines change and the aggregation is measurably too slow – maintained by a trigger (or materialised-view refresh), the mechanism that keeps it honest ✓.