Exercise Sheet: Normalization

Data, Data Storage, Data Collection

Exercise 1 – Pizzerias

A food-delivery website lists the menus of local pizzerias in a single table, where each row records one product sold by one pizzeria:

Pizzeria(Name, Address, Product, Price).

The website follows two rules: each pizzeria has a single address, and each pizzeria sets its own price for each product. Here is an extract of the table:

Name	Address	Product	Price
Fratelli	12 Main St	Margherita	9
Fratelli	12 Main St	Calzone	12
Fratelli	12 Main St	Tiramisu	6
Da Rosa	3 Park Ave	Margherita	10
Da Rosa	3 Park Ave	Diavola	11
Bella Napoli	8 Harbor Rd	Margherita	8

(**Q1**) Using rows of the table, give an example of redundancy and an example of each anomaly: insertion, update and deletion.

Redundancy: Fratelli's address is stored three times, once per product.

Update anomaly: if Fratelli moves, three rows must change. Updating only two of them leaves Fratelli with two addresses, and the table can no longer tell which one is correct.

Insertion anomaly: a new pizzeria with no menu yet cannot be stored, because Product is part of the key and a row needs one.

Deletion anomaly: if Bella Napoli stops selling its only product, deleting that row also deletes Bella Napoli's address.

The three anomalies come from the same redundancy: the address is a fact about the pizzeria, but it is stored once per product.

(**Q2**) What are the functional dependencies implied by the two rules? Show that the only candidate key is $\{\text{Name}, \text{Product}\}$.

The rules give $\{\text{Name}\} \to \{\text{Address}\}$ and $\{\text{Name}, \text{Product}\} \to \{\text{Price}\}$.

Together, $\{\text{Name}, \text{Product}\}$ determines every attribute, so it is a superkey. No dependency has Name or Product on its right-hand side, so every superkey must contain both: $\{\text{Name}, \text{Product}\}$ is minimal, and it is the only candidate key.

Other dependencies hold on these six rows, such as $\{\text{Address}\} \to \{\text{Name}\}$, but no rule implies them.

Direction. Read $X \to Y$ as "once X is known, Y is known": the left-hand side is what you look up, the right-hand side is what you get back. "Each pizzeria has a single

address": given a name, there is one address, so Name is on the left. The dependency {Address} → {Name} would say that each address hosts a single pizzeria.

(**Q3**) Determine the highest normal form of the table and justify it with the definitions. Then decompose the table into tables in BCNF, check that the split is lossless, and name the foreign key.

All values are atomic, so the table is in 1NF. The prime attributes are Name and Product, so Address is non-prime, and it depends on {Name}, a proper subset of the candidate key: 2NF is violated. The table is therefore in **1NF** only.

We split along {Name} → {Address}:

Pizzeria(Name, Address) Menu(Name, Product, Price).

Normal form. In Pizzeria, the only non-trivial dependency is {Name} → {Address}, and {Name} is the key. In Menu, it is {Name, Product} → {Price}, and {Name, Product} is the key. Every left-hand side is a superkey, so both tables are in BCNF.

Lossless. The shared attribute Name is a key of Pizzeria, so each row of Menu meets exactly one address: the join on Name gives back the original table, with no spurious row.

Dependencies. Each of the two dependencies fits inside one table, so none is lost.

Foreign key. Name in Menu references Pizzeria: the DBMS refuses a product of an unknown pizzeria. The direction is forced by the definition: a foreign key points to a *key* of the other table, and Name is a key of Pizzeria but not of Menu. The constraint states that every name in Menu appears in Pizzeria. The reverse would forbid a pizzeria with no menu, which is the insertion anomaly again. In general, splitting along $X \to Y$ creates a table with key X, and the remaining table keeps X as a foreign key to it.

Anomalies. Each one is gone, because each address is now stored once:

- update: if Fratelli moves, one row of Pizzeria changes;
- insertion: a new pizzeria with no menu is one row of Pizzeria;
- deletion: deleting Bella Napoli's last product removes a row of Menu, and its row in Pizzeria keeps the address.

Exercise 2 – University Course Registration

A university stores its course registrations in a single table. The registrar follows these rules: student IDs are unique, but two students may share a name; each course is taught by a single instructor, but an instructor may teach several courses.

StudentID	StudentName	Courses	Instructors
101	Alice	Math, Physics	Dr. Smith, Dr. Johnson
102	Bob	Chemistry, Math	Dr. Brown, Dr. Smith
103	Carol	Biology, Chemistry	Dr. Green, Dr. Brown

A – First Normal Form (1NF)

(**Q1**) Why is this table not in 1NF? Transform it into 1NF.

The Courses and Instructors cells hold lists (repeating groups), not atomic values.

Worse, the link between a course and its instructor is stored only through the position of the values in the two lists: Alice's first course goes with her first instructor.

We write one row per student and course:

StudentID	StudentName	Course	Instructor
101	Alice	Math	Dr. Smith
101	Alice	Physics	Dr. Johnson
102	Bob	Chemistry	Dr. Brown
102	Bob	Math	Dr. Smith
103	Carol	Biology	Dr. Green
103	Carol	Chemistry	Dr. Brown

B – Second Normal Form (2NF)

(**Q2**) List the functional dependencies implied by the registrar's rules, and find the candidate key(s) of the 1NF table.

$\{\text{StudentID}\} \to \{\text{StudentName}\}$ and $\{\text{Course}\} \to \{\text{Instructor}\}$.

On these six rows, $\{\text{Instructor}\} \to \{\text{Course}\}$ and $\{\text{StudentName}\} \to \{\text{StudentID}\}$ also hold, but the rules say that they may fail on future rows.

No dependency has StudentID or Course on its right-hand side, so every superkey contains both. Conversely, $\{\text{StudentID}, \text{Course}\}$ determines StudentName and Instructor, so it is a superkey, and it is contained in all the others. The only candidate key is therefore $\{\text{StudentID}, \text{Course}\}$.

(**Q3**) Is the 1NF table in 2NF? If not, explain why and decompose it into 2NF.

The 1NF table is not in 2NF.

The prime attributes are StudentID and Course; StudentName and Instructor are non-prime. Both depend on a proper subset of the candidate key: StudentName on StudentID, Instructor on Course.

We decompose into three tables,

Student(<u>StudentID</u>, StudentName),
Teaching(<u>Course</u>, Instructor),
and Enrollment(<u>StudentID</u>, <u>Course</u>).

Nothing is lost: Enrollment shares StudentID with Student and Course with Teaching, and each is a key there, so the joins rebuild the 1NF table. These two shared attributes are the foreign keys of Enrollment, the table that stores the many-to-many relationship between students and courses. No attribute of the three tables depends on part of a key anymore, so they are all in 2NF.

C – Third Normal Form (3NF)

The university now records the department of each instructor in the table of courses and instructors, and adds a Statistics course. Each instructor belongs to a single department.

Course	Instructor	Department
Math	Dr. Smith	Mathematics
Statistics	Dr. Smith	Mathematics
Physics	Dr. Johnson	Physics
Chemistry	Dr. Brown	Chemistry
Biology	Dr. Green	Biology

(**Q4**) Is this table in 3NF? If not, decompose it into 3NF. Are the resulting tables in BCNF?

This table is not in 3NF.

The dependencies are {Course} → {Instructor} and {Instructor} → {Department}, a transitive dependency, and the only candidate key is {Course}. {Instructor} → {Department} is non-trivial, Instructor is not a superkey (Dr. Smith teaches two courses), and Department is not prime: 3NF is violated. The symptom is that the Mathematics department of Dr. Smith is stored twice.

We decompose into

Teaching(Course, Instructor) Staff(Instructor, Department).

Both tables are in BCNF: in each one, the only non-trivial dependency has the key on its left. The split is lossless, because the shared attribute Instructor is a key of Staff, and no dependency is lost, because each one fits inside a table. Instructor in Teaching is a foreign key referencing Staff.

The university also runs a tutoring program. Each student has a single tutor per course, and each tutor teaches a single course:

Student	Course	Tutor
Alice	Math	Tom
Bob	Math	Lea
Carol	Math	Tom
Alice	Physics	Nina
Carol	Physics	Nina

(**Q5**) Find the candidate keys of this table. Is it in 3NF? In BCNF?

The rules give {Student, Course} → {Tutor} and {Tutor} → {Course}. No dependency has Student on its right-hand side, so every key contains Student, and Student alone is not enough (Alice has two courses). Adding Course gives a superkey, and so does adding Tutor, since Tutor determines Course. The candidate keys are therefore {Student, Course} and {Student, Tutor}.

All three attributes are prime, so every dependency satisfies the 3NF condition: the table is in 3NF. But {Tutor} → {Course} is non-trivial and Tutor is not a superkey, so the table is not in BCNF. The symptom is that "Tom teaches Math" is stored twice.

(**Q6**) Split this table along the dependency that violates BCNF. What do you gain and what do you lose? Which design do you keep?

We split along {Tutor} → {Course}, into

TutorCourse(Tutor, Course) and Assignment(Student, Tutor).

Both tables are in BCNF, the split is lossless because Tutor is a key of TutorCourse, and each tutor's course is stored once. But $\{\text{Student}, \text{Course}\} \to \{\text{Tutor}\}$ now spans two tables: Assignment accepts (Bob, Tom), and Bob then has both Lea and Tom for Math.

The split is not dependency-preserving, so we have a choice between some redundancy and a rule that no single table can check. When a dependency is lost, we stay in 3NF: here, we keep the original table.

D – Fourth Normal Form (4NF)

The university also records student activities. Each student can join several clubs and play several sports, and the clubs a student joins have nothing to do with the sports they play:

StudentID	Clubs	Sports
101	Chess, Debate	Swimming, Tennis
102	Robotics, Chess	Tennis, Soccer
103	Debate, Music	Swimming, Soccer

(**Q7**) Transform this table into 1NF. Which rows must it contain for each student? Is it in BCNF?

Each club of a student appears with each of their sports: 12 rows, four per student, e.g. (101, Chess, Swimming), (101, Chess, Tennis), (101, Debate, Swimming), (101, Debate, Tennis). Keeping only (101, Chess, Swimming) and (101, Debate, Tennis) would invent a link between clubs and sports.

The only candidate key is $\{\text{StudentID}, \text{Club}, \text{Sport}\}$: no attribute or pair determines the third. With no non-trivial functional dependency, the table is in BCNF.

(**Q8**) Is this 1NF table in 4NF? If not, decompose it into 4NF.

This table is not in 4NF.

For each student, the clubs and the sports form a Cartesian product, so two rows of a student can swap their clubs and stay in the table: $\{\text{StudentID}\} \twoheadrightarrow \{\text{Club}\}$ holds. This multivalued dependency is non-trivial and StudentID is not a superkey, so 4NF is violated. The symptom: if student 101 joins Music, two rows must be inserted, one per sport.

We decompose into

$$\text{Membership}(\underline{\text{StudentID}}, \underline{\text{Club}}) \qquad \text{Practice}(\underline{\text{StudentID}}, \underline{\text{Sport}}).$$

In a table of two attributes, every multivalued dependency is trivial, so both tables are in 4NF. If student 101 joins Music, we now insert a single row.

The split is lossless, even though StudentID is a key of neither table: the join on StudentID rebuilds every combination of a student's clubs and sports, and these are exactly the rows of the table, because clubs and sports are independent.

E – Fifth Normal Form (5NF)

The university finally records final-year projects. Each row states that a student works on a topic with a supervisor. No non-trivial functional or multivalued dependency holds, so the only candidate key is the set of all three attributes, and the table is in 4NF:

Student	Supervisor	Topic
Alice	Dr. Smith	AI
Alice	Dr. Smith	Data Science
Alice	Dr. Brown	AI
Bob	Dr. Brown	AI
Carol	Dr. Smith	Data Science

(**Q9**) Project the table onto (Student, Supervisor), (Supervisor, Topic) and (Student, Topic), and join the three projections back. Do you get the original table?

A projection keeps some columns and removes the duplicate rows, which gives:

Student	Supervisor
Alice	Dr. Smith
Alice	Dr. Brown
Bob	Dr. Brown
Carol	Dr. Smith

Supervisor	Topic
Dr. Smith	AI
Dr. Smith	Data Science
Dr. Brown	AI

Student	Topic
Alice	AI
Alice	Data Science
Bob	AI
Carol	Data Science

Joining the first two on Supervisor gives each student every topic of their supervisors: the five original rows plus (Carol, Dr. Smith, AI), a spurious row. Joining the result with (Student, Topic) removes that row, because the pair (Carol, AI) is not in that projection. So we do get the original table: it is the join of its three projections, and it satisfies a *join dependency*.

On five rows, this could be a coincidence: a sixth row could break it, as in Exercise 3.

(**Q10**) The university adopts the following rule: *if a student works with a supervisor, that supervisor supervises a topic, and the student works on that topic, then the student works on that topic with that supervisor.* How can we turn the table into 5NF?

Under this rule, every row of the three-way join is a real project, so the table is always the join of its three projections. The only candidate key is the whole set $\{\text{Student}, \text{Supervisor}, \text{Topic}\}$, and none of the three contains it, so they do not share a candidate key: the table is not in 5NF.

We decompose into the three tables from (**Q9**). Two tables are not enough: each two-way join adds a spurious row, namely (Carol, Dr. Smith, AI) on Supervisor, (Alice, Dr. Brown, Data Science) on Student and (Bob, Dr. Smith, AI) on Topic.

Without the rule, the join dependency is only a coincidence of these five rows: the table would be in 5NF, and we would keep it.

Exercise 3 – A Partner's Undocumented Extract

A partner company sends us an extract of one of its tables for a data integration project. The column names were anonymized, so we only know that the schema is $R(A, B, C, D)$, and we receive no documentation and no rules of the domain, only these five rows:

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b1	c3	d3
a2	b1	c4	d3
a3	b2	c5	d1

Recall that $X \to Y$ holds in a table if any two rows that agree on X also agree on Y. We look for the dependencies that hold on these five rows.

(**Q1**) Which single attributes determine another attribute? Which pairs of attributes determine a third one?

Each value of C appears in a single row, so two rows never agree on C, and the definition holds for $\{C\} \to \{A\}$, $\{C\} \to \{B\}$ and $\{C\} \to \{D\}$. For the other attributes, one pair of rows is enough to refute every dependency:

- A: rows 1 and 2 share a1 but differ on B, C and D;
- B: rows 1 and 3 share b1 but differ on A, C and D;
- D: rows 1 and 5 share d1 but differ on A, B and C.

A pair containing C determines every attribute, since C alone does. This leaves the pairs $\{A, B\}$, $\{A, D\}$ and $\{B, D\}$.

To check $\{A, B\} \to \{D\}$, we group the rows by their values of (A, B), and compare the values of D inside each group:

$$(a1, b1)\text{: row 1,} \quad (a1, b2)\text{: row 2,} \quad (a2, b1)\text{: rows 3, 4,} \quad (a3, b2)\text{: row 5.}$$

Only rows 3 and 4 fall in the same group, and they share d3, so $\{A, B\} \to \{D\}$ holds.

Grouping by (A, D) or by (B, D) also leaves rows 3 and 4 as the only group of two rows. They share b1 and a2, so $\{A, D\} \to \{B\}$ and $\{B, D\} \to \{A\}$ hold. None of these pairs determines C, since rows 3 and 4 differ on C.

So only C determines other attributes on its own, and the pairs without C give $\{A, B\} \to \{D\}$, $\{A, D\} \to \{B\}$ and $\{B, D\} \to \{A\}$.

(**Q2**) Deduce the candidate key(s) of R.

C determines every attribute, so $\{C\}$ is a superkey. Any set without C fails, because rows 3 and 4 agree on everything except C. So every superkey contains C, and the only candidate key is $\{C\}$.

(**Q3**) A sixth row arrives: $(a1, b1, c6, d2)$. Which dependencies of (**Q1**) still hold? What does this say about dependencies found on a sample?

The pair $(a1, b1)$ now meets both d1 (row 1) and d2 (row 6), so $\{A, B\} \to \{D\}$ breaks. The pair $(a1, d2)$ meets both b2 (row 2) and b1 (row 6), so $\{A, D\} \to \{B\}$ breaks. The pair $(b1, d2)$ is new, so $\{B, D\} \to \{A\}$ survives, and so do the dependencies on C (c6 is a new value).

A dependency found on a sample only describes the rows we have seen: the rows can refute a dependency, but only the domain can assert it.