Lab 7

#1

student_id	student_name	course
1	Ali	CS101
1	Ali	CS102
1	Ali	CS103
2	Merey	CS101
2	Merey	CS103
3	Sagibek	CS102
3	Sagibek	CS103
3	Sagibek	CS104

Explanation

The original table had a multivalued attribute ("courses") containing several course codes separated by commas.
To achieve 1NF, I decomposed these values so each row contains a single course per student.

#2

student_id	student_name
1	Ali
2	Merey

course_id	course_name	instructor
CS101	Intro to Databases	Dr. Smith
CS102	Data Structures	Dr. Lee
CS103	Web Development	Dr. Adams

student_id	course_id	grade
1	CS101	A
1	CS102	B+
2	CS101	A-
2	CS103	A

Explanation
In the original table, non-prime attributes (like student_name, course_name, instructor) were functionally dependent only on a subset of a possible composite key (student_id, course_id). By splitting, I removed partial dependencies:
Student attributes go into a student table.
Course attributes go into a course table.
Enrollment-related info (grade) stays in the linking table.

#3

student_id	student_name	advisor_id
1	Ali	101
2	Merey	101

3	Sagibek	102

advisor_id	**advisor_name**	**advisor_office**
101	Dr. Smith	Room 301
102	Dr. Adams	Room 405

Explanation
By splitting the table, advisor_name and advisor_office now depend solely on advisor_id, and student_name depends solely on student_id.
Any data about an advisor appears only once in the advisor table.
This ensures update, insert, and delete anomalies cannot arise from transitive dependency.

#4

course_id	**instructor**
CS101	Dr. Smith
CS102	Dr. Lee

course_id	**ta**
CS101	Ali
CS101	Merey
CS102	Sagibek
CS102	Zhanel

Explanation
course_id → instructor is a functional dependency, so store that relationship in its own table.
(course_id, ta) is the composite key for the set of course TA pairs.
This eliminates all BCNF violations, as no non-superkey determines an attribute.

#5

```
CREATE TABLE Reporting (
  student_id INTEGER,
  student_name TEXT,
  student_major TEXT,
  student_year INTEGER,
  student_gpa REAL,
  course_id INTEGER,
  course_name TEXT,
  course_credits INTEGER,
  course_dept TEXT,
  prof_id INTEGER,
  professor_name TEXT,
  professor_department TEXT,
  professor_rank TEXT,
  semester TEXT,
  grade TEXT
);
```

What redundancy might appear here?
Same student, course, and professor details are repeated in many rows for each enrollment or teaching record.

What anomalies might appear here?

Updating a value (like a student's major) in one row but not others causes inconsistency; inserting or deleting records can result in loss or duplication of important data.

How to make it better (without full BCNF)?
Partially normalize: separate basic reference data (students, courses, professors) into smaller tables, and use views or joins to support reporting; add constraints or triggers to help avoid inconsistencies.

#6
Unnormalized

```
CREATE TABLE DepartmentClubAdvisor (
  student_id INTEGER,
  student_name TEXT,
  department_id INTEGER,
  department_name TEXT,
  advisor_id INTEGER,
  advisor_name TEXT,
  advisor_office TEXT,
  clubs TEXT
);
```

When/Why:
Used for quick prototyping or importing raw data with minimal effort.
Schema is simple but stores repeating groups (clubs as a list), leading to lots of redundancy and high risk of inconsistencies.
Best for one-off data conversions, not for operational databases.

3NF

```
CREATE TABLE Departments (
  department_id INTEGER PRIMARY KEY,
  department_name TEXT
);
```

```
CREATE TABLE StudentClubs (
  club_id INTEGER PRIMARY KEY,
  club_name TEXT
);

CREATE TABLE Advisors (
  advisor_id INTEGER PRIMARY KEY,
  advisor_name TEXT,
  advisor_office TEXT
);

CREATE TABLE StudentAdvisors (
  student_id INTEGER,
  advisor_id INTEGER,
  PRIMARY KEY(student_id, advisor_id),
  FOREIGN KEY(advisor_id) REFERENCES Advisors(advisor_id)
);

CREATE TABLE StudentClubsMembership (
  student_id INTEGER,
  club_id INTEGER,
  PRIMARY KEY(student_id, club_id),
  FOREIGN KEY(club_id) REFERENCES StudentClubs(club_id)
);

CREATE TABLE Students (
  student_id INTEGER PRIMARY KEY,
  student_name TEXT,
  department_id INTEGER,
  FOREIGN KEY(department_id) REFERENCES Departments(department_id)
);
```

When/Why:
Ideal for operational databases where data integrity, consistency, and flexibility are critical.
Separate tables for departments, clubs, advisors, and links between them reduce redundancy, avoid anomalies, and ensure all updates are consistent.
Best for transactional systems and any situation needing clean, maintainable data.

Denormalized
```
CREATE TABLE StudentInfoReporting (
  student_id INTEGER,
  student_name TEXT,
  department_id INTEGER,
  department_name TEXT,
  advisor_id INTEGER,
  advisor_name TEXT,
  advisor_office TEXT,
  club_id INTEGER,
  club_name TEXT
);
```

When/Why:
Used for analytics/dashboards where query speed is most important and occasional data updates are acceptable.
Combines all related info into single rows for fast reads but at the expense of redundancy—update anomalies are possible.
Best for read-heavy reporting and business intelligence environments.

#7

course_code	semester	instructor
CS101	Fall	Dr. Smith
CS101	Spring	Dr. Smith
CS102	Fall	Dr. Adams

Why this is 3NF but not BCNF:
The table is in 3NF because every non-prime attribute ("instructor") is transitively dependent on a candidate key (it only depends on "course_code", which is part of a key, and not on just "semester"). However, course_code → instructor is a functional dependency where "course_code" is not a superkey, so this violates BCNF.

course_code	instructor
CS101	Dr. Smith
CS101	Dr. Smith
CS102	Dr. Adams

course_code	semester
CS101	Fall
CS101	Spring
CS102	Fall

Trade-offs
3NF Only: Fewer joins and possibly simpler queries, but allows redundancy and possible inconsistency (changing instructor for "CS101" in one row but not others).
BCNF: Removes redundancy and update anomalies by separating facts, but more join operations are needed when querying data about specific course offerings and their instructors.