Normalisation Exercise Candidate number: 97523

Schema 1
1. The schema has the following candidate key, ei the minimal attribute(s) that uniquely determine the row alone: (stuId, unit)

The stuId alone is insufficient to determine the row, because each student takes from 1-to-many classes, so it is not possible to know which class will be in a row just by knowing the student. Similarly, the unit attribute is also insufficient by itself, because many different students may enroll in each unit.

In this case, the candidate key (stuId, unit) is a composite key because neither may act as a primary key alone.

2. The Key attributes are: stuId, unit

The non-key attributes are name, gender and grade, as none of these attributes are part of the candidate key.

3. Normal form: **schema 1 is in 1NF** - my reasoning is outlined below:

To begin, I will first look at the non-trivial functional dependencies of the schema:

{stuId, unit} → grade

{stuId} → name

{stuId} → gender

(no dependencies)
{studId} -/-> unit, grade
{name} -/-> studId, gender, grade, unit
{gender} -/-> stuId, name, unit, grade
{grade} -/-> stuId, name, gender, unit

Right away, it clear there are some problems, since the composite primary key is (stuId, unit), yet there are functional dependencies outside of this paring (such as {stuId}→name). This issue will be discusse din more detail below, with normal forms.

Next, the schema is poorly designed and suffers from anomalies. These are:

- **Insert anomaly**: Because the table contains information about students, units, and enrollment / grades, there are issues when a row needs to be added that does not contain all the information of a row (ie it can't be added or there will be null values). As an example, a unit cannot be added that does not have a student enrolled in it (as might occur when a new unit is added to a course). Further, a student may not be added who is not enrolled in a class or who does not have a grade yet for class, such as at the start of the term.

- **Update anomaly**: With this schema, changing or updating certain bits of information will require updating multiple rows. So, if a unit name is changed, the unit name will need to be changed for each and every student enrolled in this unit. This is highly inefficient, as it shows that data is duplicated (wasting space) and also this sort of repetition leaves room for errors. Additionally, if a student changes their personal information (name, gender) then this information will need to be updated for each unit they are taking.
- **Delete anomaly**: In certain cases in this schema, removing a row may completely remove other, independent information from the table. An example would be removing the last student enrolled in a particular unit (such as during graduation) - doing so would also loose the name of the unit itself. Conversely, removing a unit that happens to be the only unit a student is enrolled in will also remove that student, and all their personal information, from the database.

Finally, I will work over each normal form in turn until finding where the schema is.

1NF - The schema 1 satisfies 1NF because each cell contains only one value (no sets or lists). If it didn't, for example, the relationship between a unit and the students enrolled in said unit might have been expressed (wrongly) as a single row per unit and a list of students in one cell to show all students enrolled in it.

2NF – Schema 1 does NOT satisfy 2NF because while the composite primary key is {stuId, unit}, there exists a non-trivial dependency of {stuId} → name and {stuId} → gender. In this case, attributes name and gender depend on only *part* of the primary key. So while both name and gender are non-key attributes, the definition of 2NF is still not met because XS→ A (A in this case being name, gender and XS being stuId), stuId is a key attribute not a superkey.

Because the schema fails 2NF, it automatically fails 3NF and BCNF.

Thus, schema 1 is in 1NF.

4. To normalise: The issue with the table is that it contains facts about more than one thing, and this is causing anomalies and non-ideal functional dependencies within the table. The solution is to divide the data into different tables. There will be:

- a Student table, containing a stuId as a primary key, with name and gender
- Unit table containing a unitId and a unit name
- A join table, Enrollment, containing stuId and unitId as foreign keys serving as a composite primary key. It also contains the grade each student received for the unit.

The join table is needed because of the many-to-many relationship between Student and Unit, as each student may take 1-many classes and each unit may be taken by 1-many students.

Please see the normalisation create table:

```
--normalisation for Schema 1
DROP TABLE IF EXISTS Enrollment;
DROP TABLE IF EXISTS Student;
DROP TABLE IF EXISTS Unit;

CREATE TABLE Student (
	stuId INTEGER PRIMARY KEY,
	name VARCHAR(100) NOT NULL,
	gender CHAR(1) NOT NULL,
	CONSTRAINT CHK_Gender CHECK (gender = M OR gender = F OR gender = X)
);

CREATE TABLE Unit (
	unitId INTEGER PRIMARY KEY,
	name VARCHAR(100) NOT NULL
);

CREATE TABLE Enrollment (
	stuId INTEGER NOT NULL,
	unitId INTEGER NOT NULL,
	grade CHAR(1) NOT NULL,
	PRIMARY KEY (stuId, unitId),
	FOREIGN KEY (stuId) REFERENCES Student(stuId),
	FOREIGN KEY (unitId) REFERENCES Unit(unitId)
);
```

Schema 2

1. The candidate key for Schema 2 is { city }, since it is the minimal attribute needed to uniquely determine the entire row.

2. The key attribute is { city }

Non-key attributes are country, pop, c_pop, capital

3. Normal form: **Schema 2 is in 2NF** – my reasoning is outlined below:

To begin, I will look at the non-trivial functional dependencies in the schema:

{ city } → country, pop, c_pop, capital

{ country } → c_pop

(not dependencies)
{ country } -/-> pop
{ pop } -/-> city, country, c_pop, capital
{ c_pop } -/-> city, country, capital

There is already an issue with the dependences, because an attribute should not have a functional dependency with a non-key attribute, as happens with {country} → c_pop. This will be discussed further in normal forms below.

The poor design of the schema suffers from anomalies:

- **Insert anomaly**: since city data and country data are contained within one table, it is not possible (without null values) to add a city that does not belong to a country.
- **Update anomaly**: If the country population changes, this information will need to be updated for every city within the country as well. This is clearly inefficient both in terms of storage space (duplicate information) and error prone for updates, if everything is not updated successfully there will be inconsistencies.
- **Delete anomaly**: If a city is removed that is the last entry for a particular country, then that country and country population data will also be lost.

Normal forms:

1NF - Schema 2 satisfies 1NF because all one-to-many relationships are spread across rows, rather than any one cell containing a list or set of data. If not, the table would have been designed (incorrectly) to have each country have a column that contains a list of its cities.

2NF – Schema 2 satisfies 2NF because even though {country} → c_pop, country is a non-key attribute so this dependency is allowed.

3NF – Schema 3 FAILS 3NF because of {country} → c_pop, because this is a dependency between non-key columns. The schema thus automatically fails BCNF.

Thus, schema 2 is in 2NF.

4. Normalisation. The issue with the table is that it contains data about more than one thing, thus more tables are needed. A better design could use two tables, one for cities and one for countries.

There is a one-to-many relationship between a city and a country, because each city belongs to exactly one country while each country can have from 0 to many cities. A country can exist without a city, yet a city cannot exist without a country. This parent-child relationship can be designed where the child (city) contains its countryId to the parent (Country).

Each country has exactly one capital city, and each capital city belongs to exactly one country. One way to express this relationship is a boolean on the city table, isCap (is capital) which is true when a city is a capital city.

```
--normalisation for Schema 2

DROP TABLE IF EXISTS Country;
DROP TABLE IF EXISTS City;

CREATE TABLE City(
	city_id INTEGER PRIMARY KEY,
	name VARCHAR(100) NOT NULL,
	population INTEGER NOT NULL,
	isCap BOOL NOT NULL,
	Country VARCHAR(100) NOT NULL,
	FOREIGN KEY (country) REFERENCES Country(name),
	CONSTRAINT CHK_pop CHECK (population >= 0)
);

CREATE TABLE Country (
	country_id INTEGER PRIMARY KEY,
	name VARCHAR(100) NOT NULL,
	co_pop INTEGER NOT NULL,
	CONSTRAINT CHK_co_pop CHECK (co_pop >= 0)
);
```