**COMP211 Database Design**
**Class Practice 5 Update Anomalies**

Student ID: _____ Name: _____ Date: _____

To understand the importance of good database design, let's start with an all-in-one, flat-file table design with some sample data to identify the problems caused by the initial design.

We have a database to keep track of each activity a student takes and the fee for that activity.

**Step 1: Create an Activities table** containing all the fields: student's name, activity and cost. Students can take more than one activity. To handle this requirement, we include a second activity and cost field. As a result, we have

Activities (student, activity1, cost1, activity2, cost2)

**Step 2: Test the table with some sample data.**

When creating sample data, you will notice that you can enter the same name for different students (*are they the same student or not?*), or different fees for the same activity (insertion anomaly)!

| student | activity1 | cost1 | activity2 | cost2 |
|---|---|---|---|---|
| Mary | Ballet | 500 | | |
| Peter | Football | 100 | Tennis | 90 |
| John | Swimming | 100 | Basketball | 50 |
| Tony | Swimming | 100 | | |
| Mary | Tennis | 100 | | |

Can you identify any other problems with the above design?

1. **Wasted storage space:** Not all students take a second activity. It doesn't seem much of a bother in this sample, but what if we are dealing with thousands and thousands of records, it matters.

2. **Insertion anomaly:**

   - What if a student wants to do a third activity and school rules allow it. The current design does not allow this operation!

   - Redundant information. If 50 students take swimming, we have to type in both the activity and its cost each time.

3. **Modification anomaly due to redundant data entry**: If the swimming fee goes up to $150, we have to go through *every* record containing swimming and modify the cost in each and every one.

4. **Deletion anomaly:** If we delete the first record of Mary taking Ballet, the fee of Ballet will be gone as well.

**Step 3: Modify the design.**

First, we need to find a way to identify each student uniquely by adding a unique primary key.

Activities (studentID, name, activity1, cost1, activity2, cost2)

| studentID | name | activity1 | cost1 | activity2 | cost2 |
|---|---|---|---|---|---|
| S001 | Mary | Ballet | 500 | | |
| S002 | Peter | Football | 100 | Tennis | 90 |
| S003 | John | Swimming | 100 | Basketball | 50 |
| S004 | Tony | Swimming | 100 | | |
| S005 | Mary | Tennis | 100 | | |

To solve the problem of different fees for the same activity, our Activities table has now been simplified to:

Activities (activity, cost)

| **activity** | **cost** |
|---|---|
| Ballet | 500 |
| Football | 100 |
| Swimming | 100 |
| Tennis | 100 |
| Basketball | 50 |

and Students (studentID, name)

| **studentID** | **name** |
|---|---|
| S001 | Mary |
| S002 | Peter |
| S003 | John |
| S004 | Tony |
| S005 | Mary |

Instead of having one record for all the activities a student takes, we create a separate record for each activity a student takes. Note how the new structure lets students take any number of activities – they are no longer limited to two.

Registration (studentID, activity)

| **studentID** | **activity** |
|---|---|
| S001 | Ballet |
| S002 | Football |
| S002 | Tennis |
| S003 | Swimming |
| S003 | Basketball |
| S004 | Swimming |
| S005 | Tennis |

Check and Analyse the results.

- No redundant information. Each activity fee is listed only once.
- No inconsistent data. There is only one place to input the price of each activity, so there is no chance of creating inconsistent data. And if there is a fee change, you simply update the cost in one place.
- No insertion anomalies. You can add a new activity to the Activities table without a student signing up for it.
- No deletion anomalies.

How to derive the final design? Normalization will guide us step by step to arrive at the final design.