**Database Anomalies**

**Anomalies** are undesirable consequences that can arise when you try to modify data in a table that has not been properly normalized (i.e., it has redundancy and dependencies that are not correctly structured). These anomalies can lead to inconsistencies and difficulties in maintaining the integrity of your data.

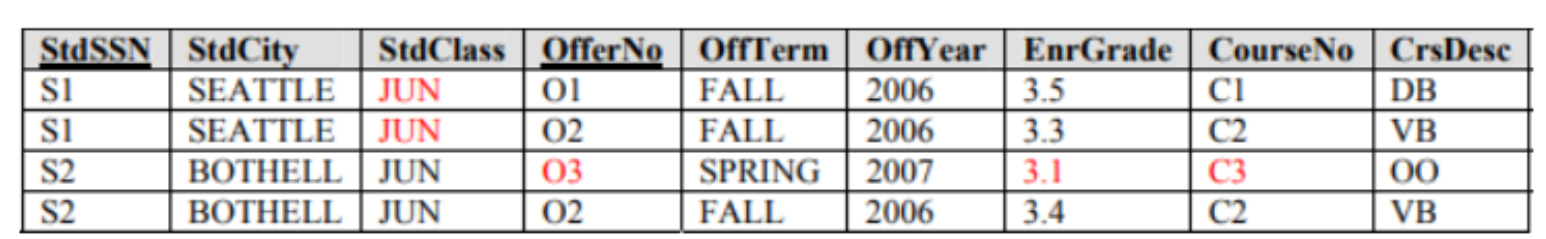
Anomalies typically arise in a relation (table) due to:

* **Data Redundancy:** The same information being stored multiple times across different rows.
* **Violation of Normal Forms:** The table not adhering to the rules of database normalization (like 1NF, 2NF, and 3NF). This often means that attributes are not grouped logically based on their dependencies on the primary key.
* **Mixing Attributes of Different Entities:** Combining information about multiple distinct entities (like staff and branches) into a single table without properly defining their relationships can lead to these issues.
* **Inappropriate Primary Keys:** Primary keys that are not minimal or that include attributes that are not truly identifiers can contribute to anomalies.
* **Transitive Dependencies:** When a non-key attribute depends on another non-key attribute rather than directly on the primary key (violating 3NF).
* **Partial Dependencies:** When a non-key attribute depends on only a part of a composite primary key (violating 2NF).

**The Goal of Normalization:**

The process of database normalization aims to restructure relations to eliminate or minimize these anomalies by:

* Reducing data redundancy.
* Ensuring that attributes are dependent only on the primary key of the relation.
* Separating information about different entities into distinct tables and establishing clear relationships between them using foreign keys.



**1. Insertion Anomaly**

This occurs when certain data **cannot be inserted** unless additional, unnecessary information is also provided.

**Example in Your Table**

* Suppose we want to **add a new course (C4: AI Systems)** to the database, but **no student has enrolled in it yet**.
* Since **CourseNo and CrsDesc** are stored within the same table **alongside student enrollment details**, we **cannot** add a new course unless at least one student is enrolled in it.
* This forces us to **create a fake enrollment record** just to store the course.

**2. Update Anomaly**

This occurs when **updating data in one place requires multiple updates**, leading to inconsistencies.

**Example in Your Table**

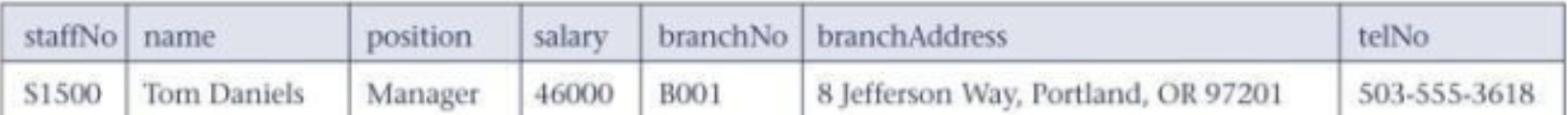
* Suppose we need to **update the course description** for **C2 (VB)** to **“Advanced VB”**.
* Since **CourseNo and CrsDesc** are repeated for multiple students, we need to **update every row** where C2 appears.
* If **one row is missed**, some students will have the outdated course name, leading to **data inconsistency**.

**3. Deletion Anomaly**

This occurs when **removing one record unintentionally deletes important information**.

**Example in Your Table**

* Suppose **Student S2 (Bothell) decides to drop all courses**.
* If we delete **S2’s records**, we **also lose information about Course C3 (OO)** unless another student is enrolled in it.
* This could lead to **loss of course information**, even though the course still exists.

**1NF:**

**Scenario 1: A Column Contains Multiple Pieces of Information (e.g., a combined address)**

If a single column holds multiple distinct pieces of information that you might need to query or manipulate individually (like the "branchAddress" column containing street, city, state, and zip code in the previous example), you should **decompose** that column into multiple atomic columns.

**Steps:**

1. **Identify the Component Parts:** Analyze the non-atomic column and determine the distinct pieces of information it contains. For "branchAddress," these would be street address, city, state, and zip code.
2. **Create New Columns:** For each identified component, create a new column in the table with an appropriate data type and name (e.g., streetAddress, city, state, zipCode).
3. **Remove the Non-Atomic Column (Optional but Recommended):** Once you've successfully migrated the data to the new atomic columns, you can remove the original non-atomic column to avoid redundancy and enforce 1NF.

**Example (from the StaffBranch Table):**

The branchAddress column is non-atomic as it contains street, city, state, and zip code. To convert this to 1NF, you would:

1. Identify components: **Street Address, City, State, Zip Code.**
2. Create new columns: streetAddress (VARCHAR), city (VARCHAR), state (VARCHAR), zipCode (VARCHAR).
3. Migrate data: Parse the branchAddress values and populate the new columns. For example, "8 Jefferson Way, Portland, OR 97201" would be split into "8 Jefferson Way", "Portland", "OR", and "97201" respectively.
4. Remove branchAddress column.

**Scenario 2: A Column Contains Multiple Values (Multi-valued Attribute)**

If a single column holds multiple distinct values for a single entity (e.g., a "phoneNumbers" **column containing 2 phone numbers for one staff** "555-055-1234, 555-365-5678"), you have a **true multi-valued attribute**, which directly violates 1NF. The standard way to address this is to **create a new, separate table** to store these multiple values.

**Steps:**

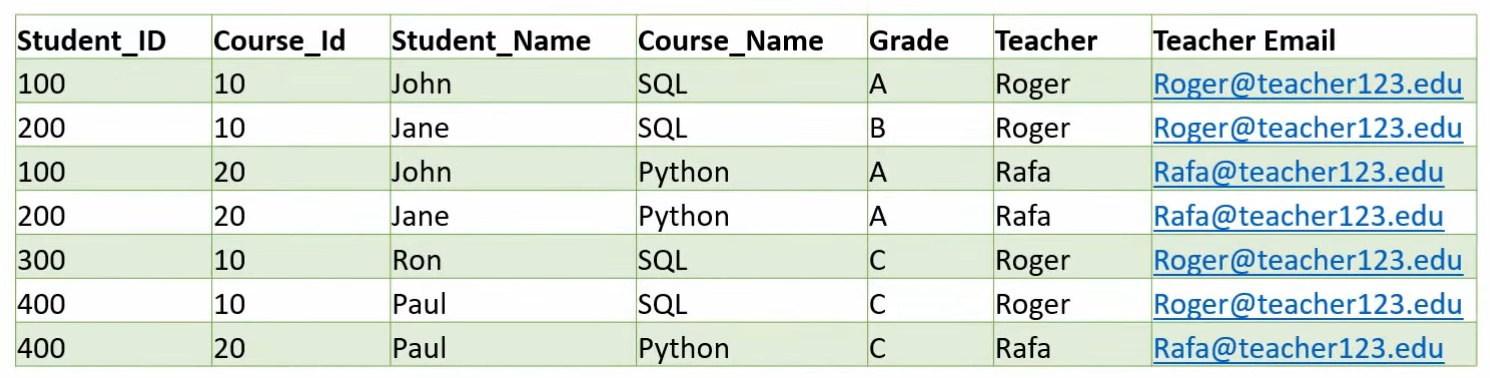
1. **Identify the Entity and the Multi-valued Attribute:** Determine the main entity of the original table and the column containing multiple values.
2. **Create a New Table:** Create a new table with at least two columns:
   * A foreign key that references the primary key of the original entity table. This establishes the relationship between the entity and its multiple values.
   * A column to store each individual value of the multi-valued attribute.
3. **Remove the Multi-valued Column from the Original Table:** Once the data is migrated, remove the non-atomic, multi-valued column from the original table.

**Example (if some Staff had multiple phone numbers):**

If the StaffBranch table had a phoneNumbers column with **comma-separated numbers**:

1. Entity: Staff, Multi-valued Attribute: phoneNumbers.
2. Create a new table: StaffPhoneNumbers with columns staffNo (FOREIGN KEY referencing StaffBranch.staffNo) and phoneNumber (VARCHAR).
3. Migrate data: For Tom Daniels (staffNo S1500) with phoneNumbers "503-555-3618, 503-555-9999", you'd create two rows in StaffPhoneNumbers:
   * staffNo: S1500, phoneNumber: 503-555-3618
   * staffNo: S1500, phoneNumber: 503-555-9999
4. Remove phoneNumbers from StaffBranch.

**Un-normalized Table:**

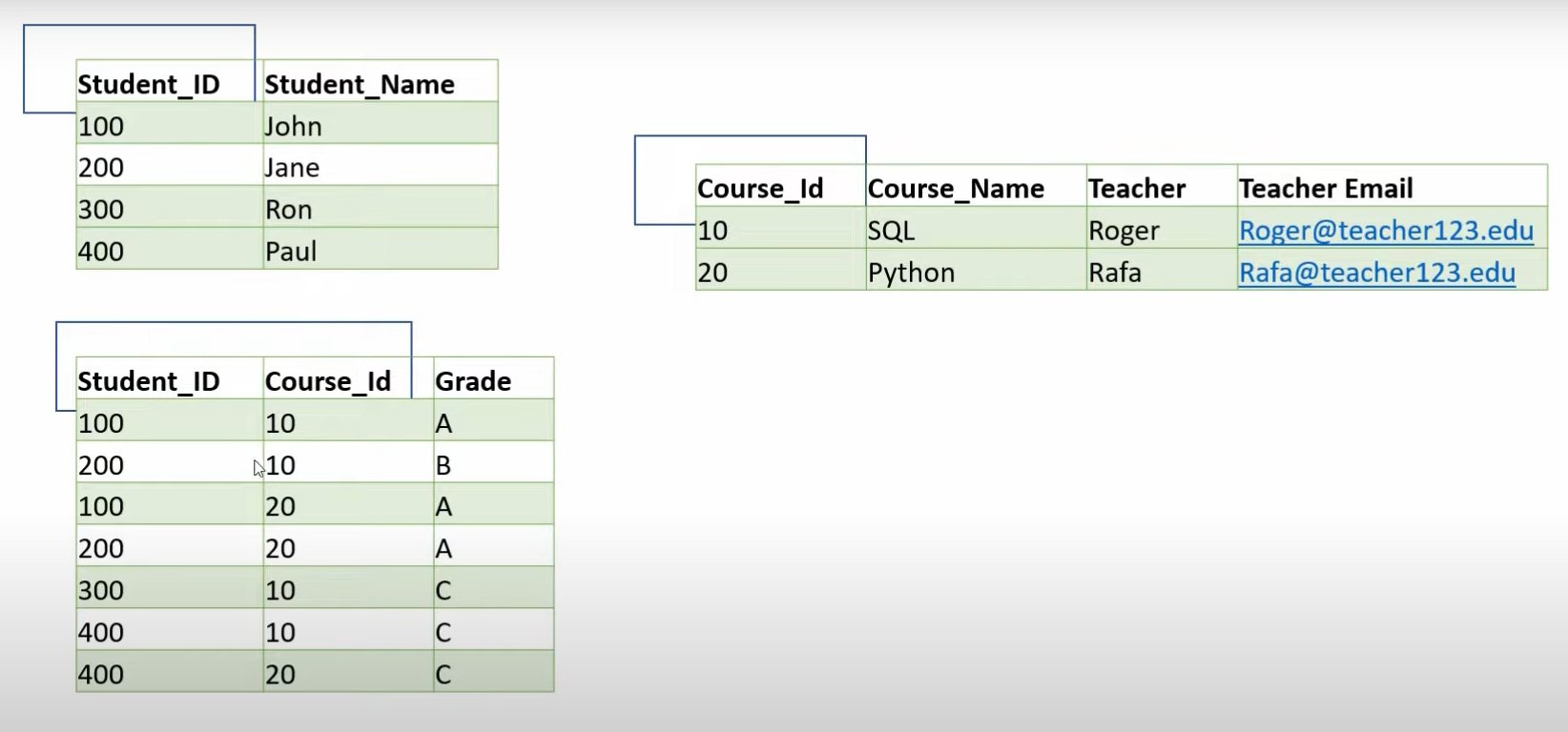


**1NF:** All columns must be atomic and unique (primary key).

* **Primary key** (Student\_ID, Course\_ID)

**2NF:** Must be in 1NF and all non-key columns must fully functionally depend on the entire primary key. If not then split the table.

* **Student** (Student\_ID, Student\_Name)
* **Course** (Course\_ID, Course\_Name, Teacher, Teacher\_Email)
* **StudentCourse** (Student\_ID , Course\_ID, Grade) **Junction table for** **M:N**



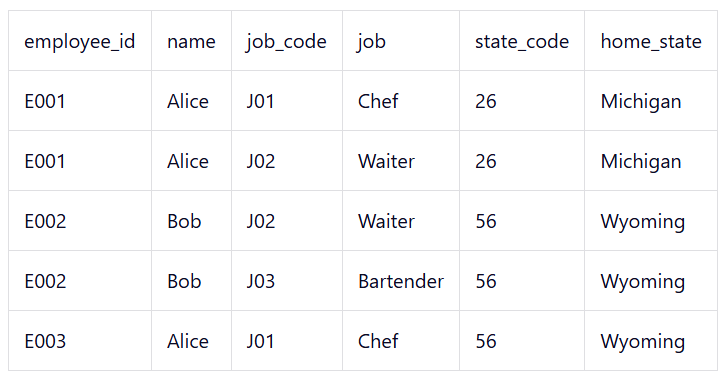
**3NF:** Must be in 2NF andall non-key attributes must be non-transitively dependent on the primary key. This means a non-key attribute should not depend on another non-key attribute.

* **Teacher** (ID, Teacher, Teacher\_Email)
* **Course** (Course\_ID, Course\_Name, Teacher\_ID (FK))

A screenshot of a computer

AI-generated content may be incorrect.

**Un-normalized Table:**

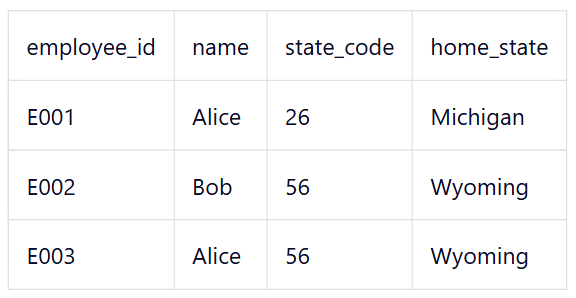


**1NF:**

* **Primary Key** (employee\_id, job\_code)

**2NF:**

* **Employee** (employee\_id, name, state\_code, home\_state)
* **Job** (job\_code, job)
* **EmployeeRole** (employee\_id, job\_code) **Junction table for** **M:N**

 A screenshot of a computer code

AI-generated content may be incorrect.

**3NF:**

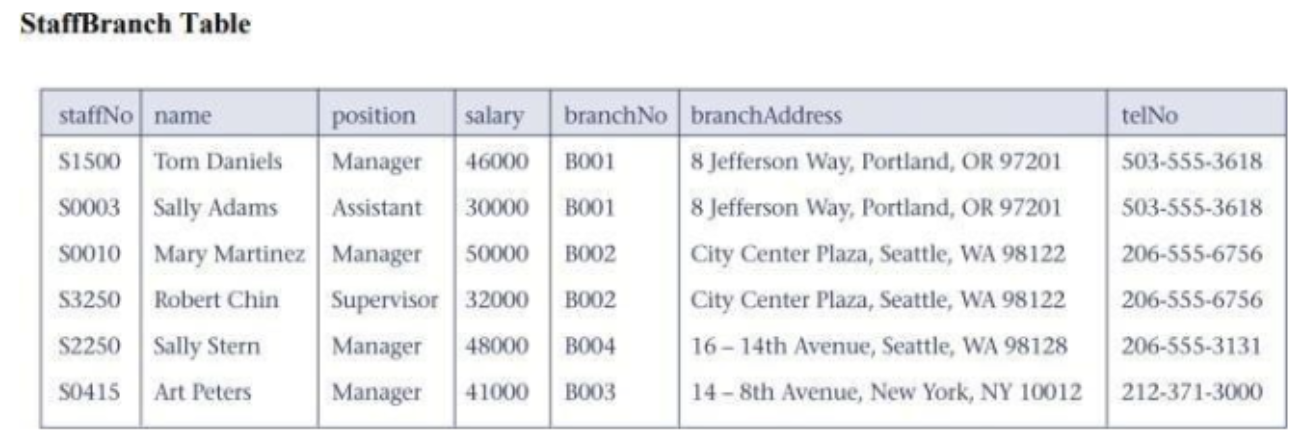
* **State** (state\_code, home\_state)
* **Employee** (employee\_id, name, state\_code (FK))

A grid of white squares with black text

AI-generated content may be incorrect. A white grid with black text

AI-generated content may be incorrect.

**Un-normalized Table:**



**1NF:**

* **Primary key** (staffNo, BranchNo)
* Decompose **branchAddress** into separate **atomic columns**: **street, city, state, zip** within the original table.
* **StaffBranch** (staffNo, name, position, salary, branchNo, street, city, state, zip, telNo)

**2NF:**

* **Staff** (staffNo, name, position, salary, branchNo (FK))
* **Branch** (branchNo, street, city, state, zip, telNo)

3NF:

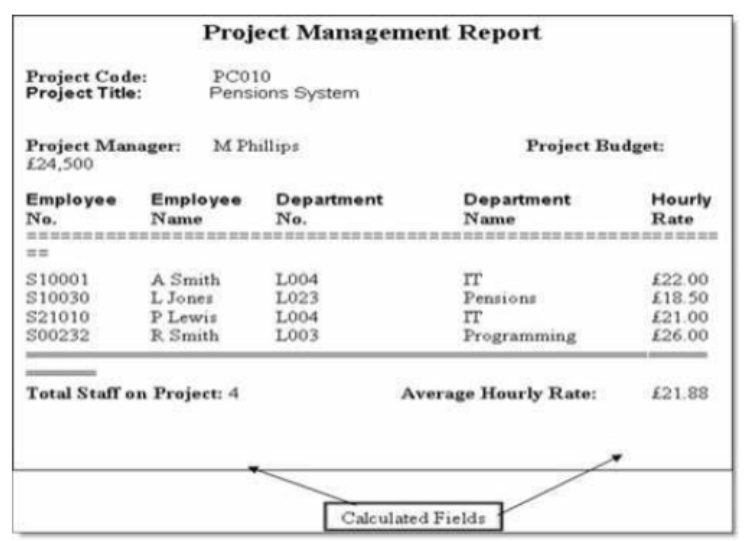
* There are **no transitive dependencies**. We've already **removed** the transitive dependency by creating a **Branch Table**.

A screenshot of a computer screen

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

**Un-normalized Scenario:**



**1NF:**

* **Primary key** (ProjectCode, EmployeeNo)
* **ProjectAssignment** (ProjectCode, ProjectTitle, ProjectManager, ProjectBudget, EmployeeNo, EmployeeName, HourlyRate, DepartmentNo, Department Name)

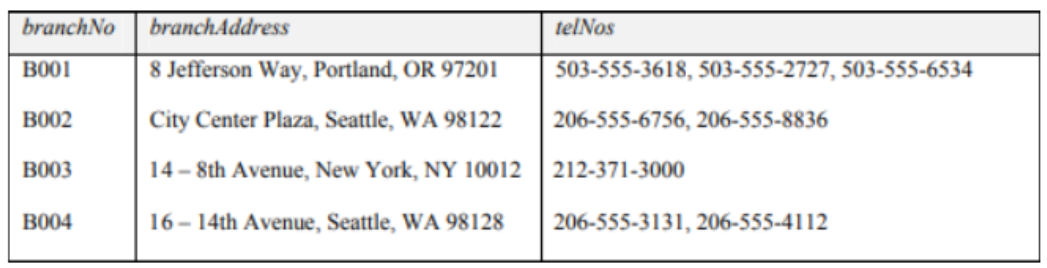
**2NF:**

* **Project** (ProjectCode, ProjectTitle, ProjectManager, ProjectBudget)
* **Employee** (EmployeeNo, EmployeeName, HourlyRate, DepartmentNo (FK))
* **Department** (DepartmentNo, DepartmentName)
* **EmployeeProject** (ProjectCode, EmployeeNo)

**3NF:**

* There are **no transitive dependencies** (i.e., all attributes depend **only on the primary key**).

**Un-normalized Table:**



**1NF:**

* **Branch** (branchNo, street, city, state, zip)
* **BranchTelephones** (branchNo, telNo)
* **BranchTelephones –** CompsitePrimary Key(branchNo, telNo)
* **BranchTelephones –** Foreign key(branchNo)

**2NF:**

* The tables are already in **2NF** since each **non-key attribute** depends **fully** on its respective **primary key**.

**3NF:**

* Since no **non-key attributes** are dependent on anything other than their **direct primary key**, the tables are already in **3NF**.



**1NF:**

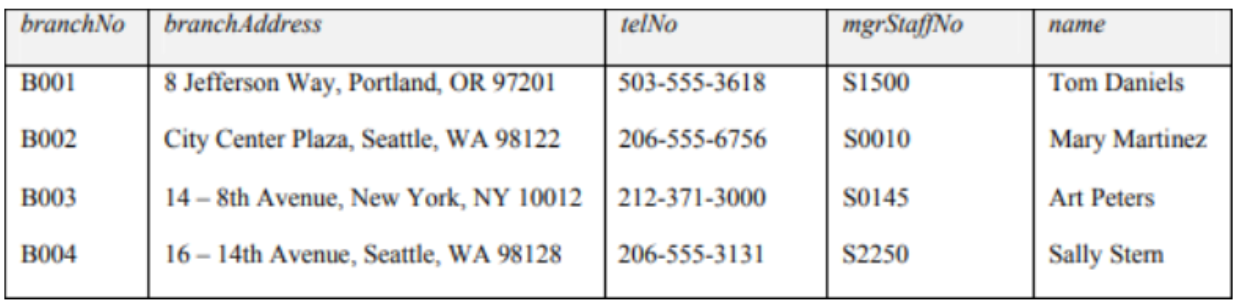
* **StaffBranch** (staffNo, branchNo, street, city, state, zip, firstName, lastName, position, hoursPerWeek)

**2NF:**

* **Staff** (staffNo, firstName, lastName)
* **Branch** (branchNo, street, city, state, zip)
* **StaffBranch** (staffNo, branchNo, position, hoursPerWeek) **Junction table for** **M:N**

**3NF:**

* There are **no transitive dependencies** (i.e., all attributes depend **only on the primary key**).



**1NF:**

* **StaffBranch** (branchNo, street, city, state, zip, telNo, mgrStaffNo, firstName, lastName)

**2NF:**

* **MgrStaff** (mgrStaffNo, firstName, lastName)
* **Branch** (branchNo, street, city, state, zip, telNo, mgrStaffNo (FK))

**3NF:**

* There are **no transitive dependencies** (i.e., all attributes depend **only on the primary key**).



**1NF:**

* **Employee** (Emp\_id, Emp\_name, Emp\_address)
* **EmployeeMobile** (Emp\_id, Emp\_mobile)

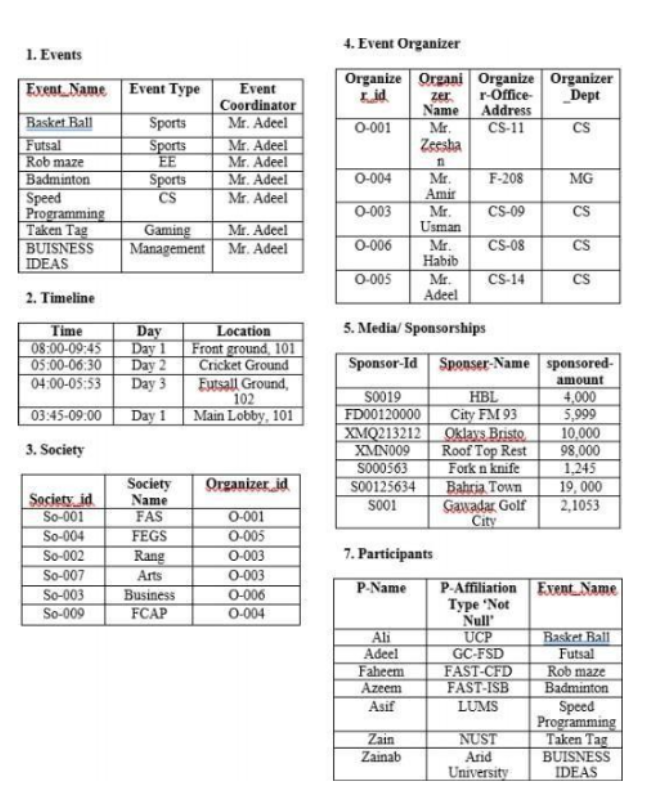
**2NF:**

* The tables are already in **2NF** since each **non-key attribute** depends **fully** on its respective **primary key**.

**3NF:**

* There are **no transitive dependencies** (i.e., all attributes depend **only on the primary key**).

**Un-normalized Database:**



**i.** **Update Anomaly: Events Table**

The **Event Coordinator**'s name (Mr. Adeel) is repeated for multiple events. If the name of the Event Coordinator for these events changes (e.g., due to a change in their title or a correction in their spelling), we would need to update this information in multiple rows – for every event coordinated by that person.

* **Events** (Event\_id, Event\_Name, Event\_Type, Organizer\_id)

Now, if any changes are required for any organizer, we only need to change it in **EventOrganizer Table.**

**ii. Impact on Anomalies: After changing data type from Not NULL to NULL for P-Affiliation in Participants Table**

**A. Insert Anomaly:**

* **Before the Change ('Not Null'):** To insert a new participant, we *must* know their affiliation. If we have information about a participant's name and the event they are participating in, but we don't know their affiliation, we cannot add them to the table.
* **After the Change (Allowing NULL):** If 'P-Affiliation' can be NULL, we *can* insert a new participant's name and the event they are in, even if their affiliation is unknown at the time of insertion. We no longer need the affiliation to record basic participation data.

**B. Delete Anomaly:**

* Delete anomalies occur when deleting a record unintentionally removes information about another entity.
* Deleting a participant's record (e.g., removing Ali from Basket Ball) only affects that specific participation record. It doesn’t affect any other data.

**C. Update Anomaly:**

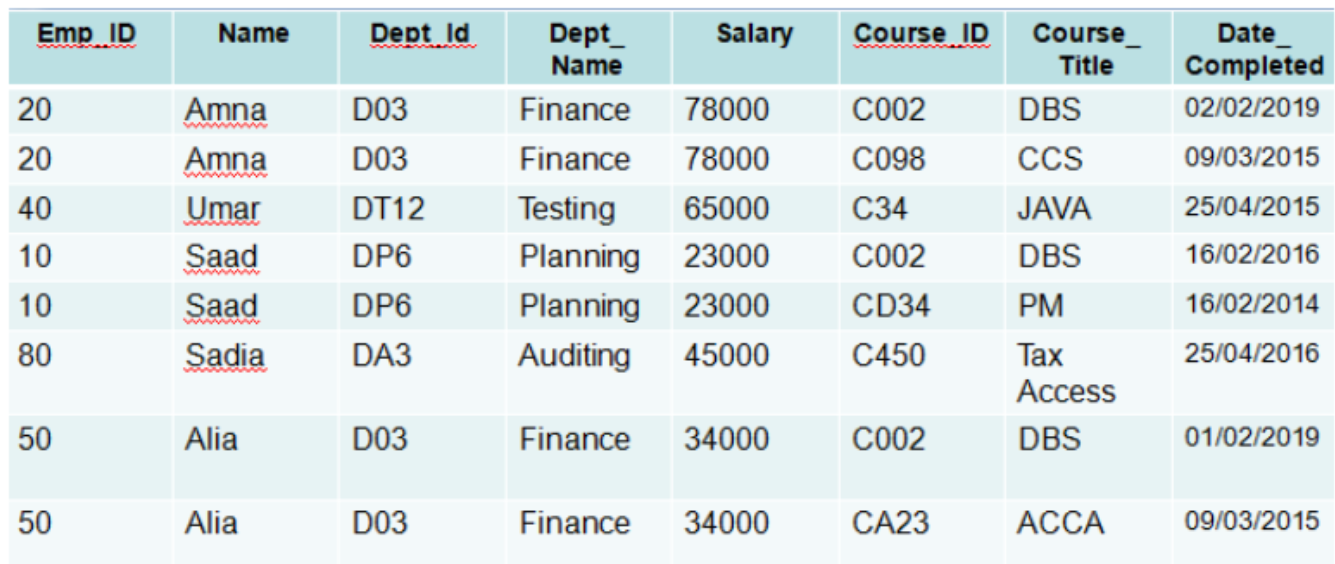
* If a participant's affiliation changes, we need to update the 'P-Affiliation' value in their corresponding record. The 'Not Null' constraint doesn't directly cause an update anomaly in this scenario. Update anomalies typically arise from data redundancy, where the same piece of information is stored in multiple places.

**iii. Delete Anomalies in the Database**

The database exhibits a **delete anomaly** in the **Event Organizer** table. Specifically:

* If an organizer (e.g., Mr. Adeel, O-005) is deleted from the **Event Organizer** table, all events coordinated by them in the **Events** table (e.g., "Futsal," "Basket Ball") will lose their coordinator information.
* This happens because the **Event Coordinator** field in the **Events** table is directly dependent on the **Organizer Name** in the **Event Organizer** table without proper foreign key constraints or normalization.

**Un-normalized Database:**



**1NF:**

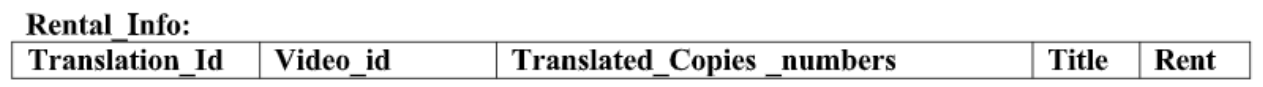
* **EmployeeDetails** (Emp\_ID, Name, Dept\_ID, Dept\_Name, Salary, Course\_ID, Course\_Title, Date\_Completed)

**2NF:**

* **Employee** (Emp\_ID, Name, Salary, Dept\_ID (FK))
* **Department** (Dept\_ID, Dept\_Name)
* **Course** (Course\_ID, Course\_Title)
* **EmployeeCourse** (Emp\_ID, Course\_ID, Date\_Completed) **Junction table for** **M:N**

**3NF:**

* There are **no transitive dependencies** (i.e., all attributes depend **only on the primary key**).



**1NF:**

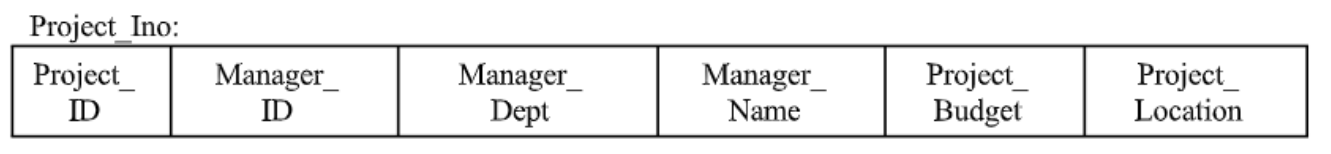
* **Rental\_Info** (Translation\_ID, Video\_ID, Number\_Translated\_Copies, Title, Rent)

**2NF:**

* **Video** (Video\_ID, Title)
* **Translation** (Translation\_ID, Video\_ID (FK), Number\_Translated\_Copies, Rent)

**3NF:**

* There are **no transitive dependencies** (i.e., all attributes depend **only on the primary key**).



**1NF:**

* **ProjectInfo** (Project\_ID, Manager\_ID, Manager\_Dept, Manager\_Name, Project\_Budget, Project\_Location)

**2NF:**

* **Project** (Project\_ID, Budget, Location, Manager\_ID (FK))
* **Manager** (Manager\_ID, Name, Dept\_ID (FK))
* **Department** (Dept\_ID, Dept\_Name)

**3NF:**

* There are **no transitive dependencies** (i.e., all attributes depend **only on the primary key**).