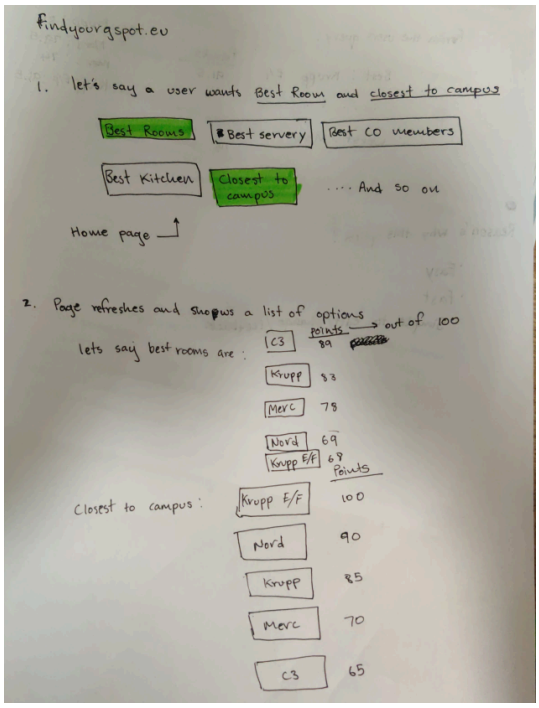

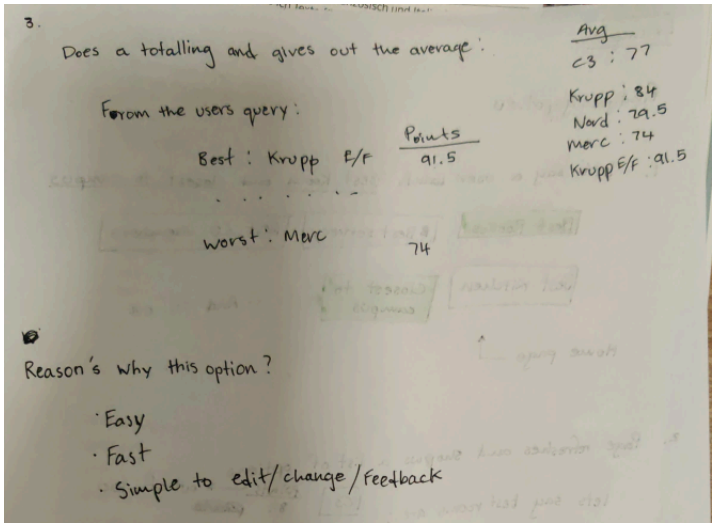

For this assignment, four entity sets and two relationship sets are implemented:

- **Entities**: `Students`, `Blocks`, `Preferences`, `Managers`
- **Relationships**: `Lives_in`, `Prefers`

These entity sets and relationships are linked in the UML diagram and are the focus of the input pages for this project.

The entities we have used in this example are different from the diagram above but it's much easier to understand using these entities because we have relationships with the same name.

2. Input Pages:

The input pages are constructed for each entity and relationship set to allow users to choose data as something like a MCQ box which the user can choose whichever he/she wants and the highest points according to the user's search comes on top. Each input page is accompanied by a feedback page that informs the user of success or errors during the operation.

Entity Input Pages

Page 1: Input Form for `Students` Entity

The student entity includes details such as `student_id`, `name`, and other attributes, excluding the `student_id`, which will be automatically generated by the server.

- **Form fields**:
  - Name: Text input for the student's name.

  HTML Example:

```
<form action="submit_student.php" method="POST">
    <label for="student_name">Name:</label><br>
    <input type="text" id="student_name" name="student_name" required><br><br>
    <input type="submit" value="Submit">
</form>
```

-

Page 2: Input Form for `Blocks` Entity

The block entity requires input for the block's letter (`letter`), and a `cleanliness` status (`is_clean`), a boolean value indicating if the block is clean or not.

- **Form fields**:
  - Block Letter: Text input.
  - Cleanliness Status: Checkbox input for whether the block is clean.

  HTML Example:

```
<form action="submit_block.php" method="POST">
```

```
    <label for="block_letter">Block Letter:</label><br>

    <input type="text" id="block_letter" name="block_letter"
required><br><br>

    <label for="is_clean">Is Clean:</label>

    <input type="checkbox" id="is_clean" name="is_clean"><br><br>

    <input type="submit" value="Submit">

</form>
```

-

Page 3: Input Form for `Preferences` Entity

The preferences entity will include attributes such as `preference_id`, `social_life`, and `distance`. Again, `preference_id` is generated server-side.

Form fields:

- Social Life Preference: Checkbox input to indicate if the student values social life.
- Distance Preference: Numeric input to capture the preferred distance.

HTML Example:

```
<form action="submit_preference.php" method="POST">

    <label for="social_life">Prefers Social Life:</label>

    <input type="checkbox" id="social_life"
name="social_life"><br><br>

    <label for="distance">Preferred Distance:</label><br>

    <input type="number" id="distance" name="distance"
required><br><br>

    <input type="submit" value="Submit">

</form>
```

Page 4: Input Form for `Managers` Entity

The manager entity includes attributes such as `manager_id` and `is_strict`. The `manager_id` is generated server-side.

- **Form fields**:
  - Strict Manager: Checkbox input to specify if the manager is strict.

  HTML Example:

```
<form action="submit_manager.php" method="POST">

    <label for="is_strict">Is Strict:</label>

    <input type="checkbox" id="is_strict"
name="is_strict"><br><br>

    <input type="submit" value="Submit">

</form>
```

Relationship Input Pages

Page 5: Input Form for `Lives_in` Relationship

The `Lives_in` relationship connects `Students` and `Blocks`. A dropdown menu is provided for both the student and the block, allowing the user to select from available entities.

- **Form fields**:
  - Student ID: Dropdown populated from the `Students` table.
  - Block ID: Dropdown populated from the `Blocks` table.

  HTML Example:

```
<form action="submit_lives_in.php" method="POST">

    <label for="student_id">Student:</label>

    <select id="student_id" name="student_id">

        <!-- Options dynamically populated from Students table
-->
```

```
    </select><br><br>

    <label for="block_id">Block:</label>

    <select id="block_id" name="block_id">

        <!-- Options dynamically populated from Blocks table -->

    </select><br><br>

    <input type="submit" value="Submit">

</form>
```

Page 6: Input Form for `Prefers` Relationship

The `Prefers` relationship connects `Students` with `Preferences`. A dropdown menu is provided for both the student and the preference.

- **Form fields**:
  - Student ID: Dropdown populated from the `Students` table.
  - Preference ID: Dropdown populated from the `Preferences` table.

  HTML Example:

```
<form action="submit_prefers.php" method="POST">

    <label for="student_id">Student:</label>

    <select id="student_id" name="student_id">

        <!-- Options dynamically populated from Students table
-->

    </select><br><br>

    <label for="preference_id">Preference:</label>

    <select id="preference_id" name="preference_id">

        <!-- Options dynamically populated from Preferences table
-->
```

```
    </select><br><br>

    <input type="submit" value="Submit">

</form>
```

3. Feedback Pages:

Each input form will redirect to a feedback page, confirming the success or failure of the operation. This feedback page will include a link back to the central maintenance page, allowing the user to navigate back to the main menu.

4. Central Maintenance Page:

A central page is created to maintain all entity and relationship input forms. This page contains links to each form and serves as the central navigation hub.

HTML Example:

```
<h1>Central Maintenance Page</h1>

<ul>

    <li><a href="student_form.html">Add New Student</a></li>

    <li><a href="block_form.html">Add New Block</a></li>

    <li><a href="preference_form.html">Add New
Preference</a></li>

    <li><a href="manager_form.html">Add New Manager</a></li>

    <li><a href="lives_in_form.html">Assign Student to
Block</a></li>

    <li><a href="prefers_form.html">Assign Preference to
Student</a></li>

</ul>
```

5. Ranking Query Implementation:

The ranking query allows users to select options, and based on those selections, the system shows results from highest points to lowest.

Example SQL for Ranking Dorms:
sql
Copy code

```
SELECT B.letter, B.is_clean, B.distance_to_class

FROM Blocks B

WHERE B.is_clean = TRUE
```

- `ORDER BY B.distance_to_class ASC;`

QUERIES FOR REFERENCE FROM ASSIGNMENT 3

Query 1: Find all students who prefer living in clean places and their block information.

Natural Language Phrasing: Retrieve all students who prefer living in clean places, along with the blocks they live in.

```
SELECT S.name, B.letter AS BlockLetter, B.is_clean
FROM Students S
JOIN Prefers P ON S.student_id = P.student_id
JOIN Preferences PR ON P.preference_id = PR.preference_id
JOIN Lives_in L ON S.student_id = L.student_id
JOIN Blocks B ON L.block_id = B.block_id
WHERE PR.is_clean = TRUE;
```

Explanation:

- This query joins `Students`, `Prefers`, `Preferences`, `Lives_in`, and `Blocks` tables.
- It selects students who prefer clean places (`PR.is_clean = TRUE`) and retrieves the block they live in (`B.letter`).

Query 2: Find the number of students in each block, grouped by the cleanliness of the block.

Natural Language Phrasing: Count how many students live in each block, grouped by whether the block is clean or not.

```
SELECT B.letter AS BlockLetter, B.is_clean, COUNT(L.student_id) AS TotalStudents
FROM Blocks B
JOIN Lives_in L ON B.block_id = L.block_id
GROUP BY B.letter, B.is_clean;
```

Explanation:

- The query joins the `Blocks` and `Lives_in` tables to count how many students live in each block.
- The results are grouped by the block (`B.letter`) and the cleanliness status of the block (`B.is_clean`).

Query 3: Find the average distance students prefer for their social life preferences.

Natural Language Phrasing: Get the average preferred distance for students who have a social life preference.

```
SELECT AVG(PR.distance) AS AverageDistance
FROM Preferences PR
WHERE PR.social_life = TRUE;
```

Explanation:

- This query selects the average distance from the `Preferences` table for students who prefer social life (`PR.social_life = TRUE`).

Query 4: List blocks where the average friendliness level of blockmates is above 7.

Natural Language Phrasing: Show the blocks where the average friendliness level of the blockmates is greater than 7.

```
SELECT B.letter AS BlockLetter, AVG(BM.friendliness_level) AS AvgFriendliness
FROM Blocks B
JOIN Blockmate BM ON B.block_id = BM.block_id
```

```
GROUP BY B.letter
HAVING AVG(BM.friendliness_level) > 7;
```

Explanation:

- This query joins `Blocks` and `Blockmate`, then calculates the average friendliness level for each block (`AVG(BM.friendliness_level)`).
- The `HAVING` clause filters blocks where the average friendliness level is greater than 7.

Query 5: Find the total number of blocks managed by strict managers.

Natural Language Phrasing: Count how many blocks have a strict manager.

```
SELECT COUNT(DISTINCT W.college_id) AS TotalManagedBlocks
FROM Manager M
JOIN Works_for W ON M.manager_id = W.manager_id
WHERE M.is_strict = TRUE;
```

Explanation:

- This query counts the total number of blocks (`COUNT(DISTINCT W.college_id)`) that have strict managers (`M.is_strict = TRUE`).
- The query joins the `Manager` and `Works_for` tables to match managers with the colleges they manage.

Query 6: Find the most active block (highest activity level).

Natural Language Phrasing: Retrieve the block with the highest activity level.

```
SELECT B.letter AS BlockLetter, B.activity_level

FROM Blocks B

ORDER BY B.activity_level DESC

LIMIT 1;
```

- **Explanation:**

- This query retrieves the block with the highest activity level by ordering blocks in descending order (`ORDER BY B.activity_level DESC`) and limiting the result to one row (`LIMIT 1`).

Query 7: List all colleges with more than 2 blocks and the total number of blocks they have.

Natural Language Phrasing: Find all colleges that have more than 2 blocks, along with the total number of blocks for each college.

SELECT C.name AS CollegeName, COUNT(B.block_id) AS TotalBlocks

FROM Colleges C

JOIN Blocks B ON C.college_id = B.college_id

GROUP BY C.name

HAVING COUNT(B.block_id) > 2;

Explanation:

- The query joins the `Colleges` and `Blocks` tables, counts the number of blocks per college (`COUNT(B.block_id)`), and filters colleges with more than 2 blocks using the `HAVING` clause.

Query 8: Find all managers who manage a college and also live in the same college block.

Natural Language Phrasing: Retrieve managers who both manage a college and live in a block within that college.

SELECT M.manager_u, S.name AS ManagerName, C.name AS CollegeName

FROM Manager M

JOIN Works_for WF ON M.manager_id = WF.manager_id

JOIN Colleges C ON WF.college_id = C.college_id

JOIN Lives_in L ON M.student_id = L.student_id

JOIN Blocks B ON L.block_id = B.block_id

WHERE B.college_id = WF.college_id;

Explanation:

- This query joins `Manager`, `Works_for`, `Colleges`, `Lives_in`, and `Blocks` to find managers who both manage a college and live in a block within that college (`B.college_id = WF.college_id`).

Query 9: Find the total number of students living in blocks that are not clean.

Natural Language Phrasing: Count how many students live in blocks that are not clean.

SELECT COUNT(L.student_id) AS TotalStudents

FROM Lives_in L

JOIN Blocks B ON L.block_id = B.block_id

WHERE B.is_clean = FALSE;

Explanation:

- This query counts the total number of students (`COUNT(L.student_id)`) who live in blocks that are not clean (`B.is_clean = FALSE`).