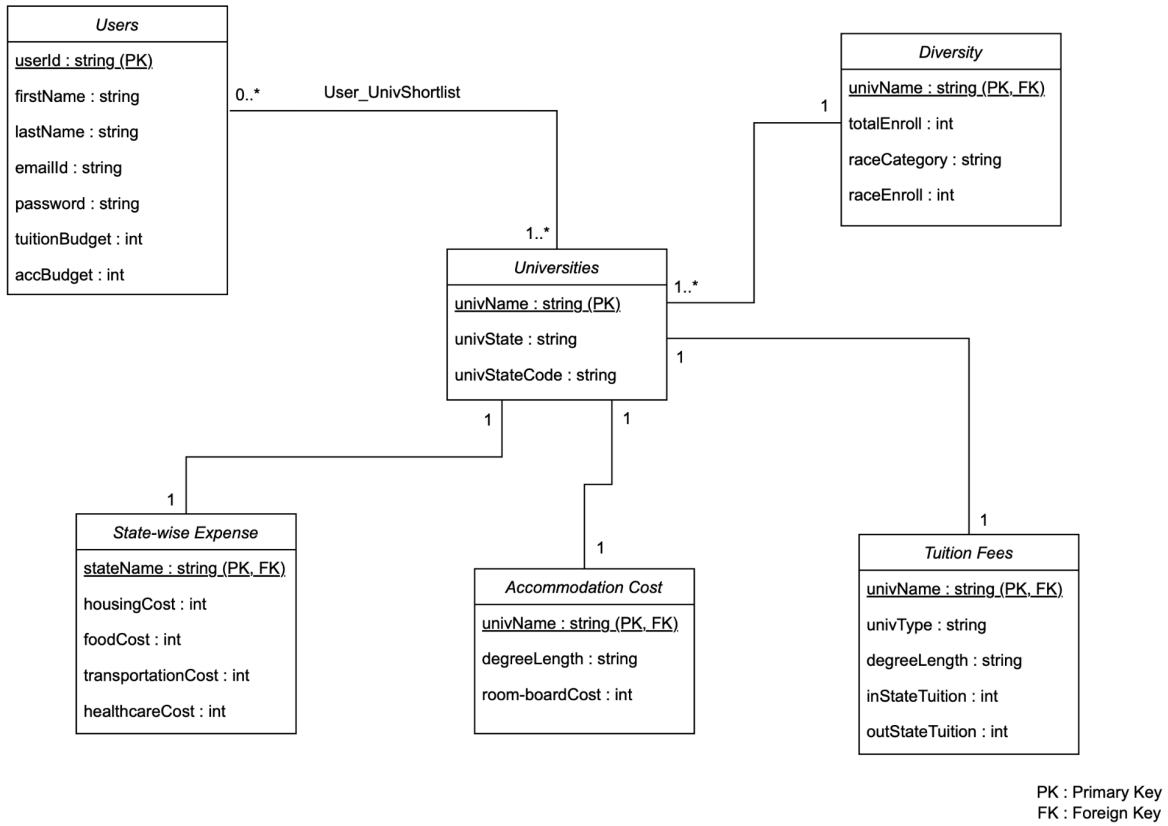
Stage 2: Database Design

Project Title: GlobalScholar: International Student Financial Navigator

Group Members: Neel Harip(nhari7), Sakshil Verma(sakshil2),
Sejal Pekam(spekam2), Sharvari Gadiwan(sgadi5)

1. UML Diagram



UML Diagram - GlobalScholar: International Student Financial Navigator

2. Explain your assumptions for each entity and relationship in your model. Discuss why you've modeled something as an entity rather than an attribute of another entity. Describe the cardinality of relationships, like why a student is linked to only one advisor. These assumptions might come from customer requirements or application constraints. Please clarify them.

1. User Entity:

- **Attributes:** Id, First Name, Last Name, Email Id, Password, list of universities, Tuition fee budget, Accommodation budget.
- **Assumptions:** The User entity represents students using the platform, who create profiles to search for universities. Each user is uniquely identified by Id and may have preferences like tuition fee budget and accommodation budget.
- **Why Modeled as an Entity:** The user has multiple dynamic attributes (e.g., preferences, budgets, and login details) that require secure storage and easy retrieval. Modeling the user as an entity supports personalization and scalability for managing individual profiles.
- **Relationships:**
    - **Many-to-Many with University** (via **User_Univ**): Each user can be interested in multiple universities, and each university can have multiple users linked to it.

2. University Entity:

- **Attributes:** University Name, State,State Code.
- **Assumptions:** The University entity represents different universities. Each university is uniquely identified by its University Name, and it also includes the State where it is located. This entity stores static information about universities and is linked to other entities to provide users with more detailed data such as tuition fees, diversity, and state-wise expenses.
- **Why Modeled as an Entity:** The university is a core component of the platform, and users need to evaluate multiple universities based on various criteria (e.g., tuition fees, accommodation). Storing this information in an entity allows each university to be associated with diverse data such as tuition, accommodation, and state-wise expenses.
- **Relationships:**
    - **One-to-Many with Diversity:** Each university has one diversity profile, but this profile contains multiple race categories and enrollment numbers.

- **One-to-One with Tuition Fees:** Each university has a unique tuition fee structure based on degree length and state residency (in-state and out-of-state tuition).
- **One-to-One with State-wise Expenses:** The state-wise expenses related to housing, food, and transportation are associated with universities based on the state they are located in.
- **Many-to-Many with User** (via **User_Univ**): Each university can be associated with multiple users.

3. Diversity Entity:

- **Attributes:** University Name, Total Enrollment, Race Category, Race-wise Enrollment.
- **Assumptions:** The Diversity entity represents universities' diversity profiles, including race-wise enrollment data.
- **Why Modeled as an Entity:** Diversity data varies widely across universities and needs to be updated periodically. Treating it as an entity allows detailed querying based on race-wise enrollment and keeps the university entity focused on core attributes.
- **Relationships:**
  - **Many-to-One with University:** Each university has only one diversity profile, but this profile contains multiple race categories and enrollment numbers.

4. State-wise Expense Entity:

- **Attributes:** State, Housing Cost, Food Cost, Transportation Cost, Healthcare Cost
- **Assumptions:** This entity contains state-specific expenses that help users calculate the cost of living for universities in different states.
- **Why Modeled as an Entity:** Living expenses differ by state, and having a separate entity allows accurate cost estimates for students evaluating universities. This data is linked to universities based on their location, making it crucial for financial planning.
- **Relationships:**
  - **One-to-One with University:** Each university is located in a specific state, and therefore it inherits the cost-of-living data for that state.

5. Accommodation Entity:

- **Attributes:** University Name, Degree Length, Room and Board Cost.
- **Assumptions:** This entity logs user searches based on accommodation costs and degree length.
- **Why Modeled as an Entity:** Accommodation costs are dynamic and differ per user and university. Storing accommodation data separately allows tracking of searches and supports the comparison of room and board costs.
- **Relationships:**
    - **One-to-One with University:** Each university has specific accommodation-related costs for room and board based on degree length.

6. Tuition Fees Entity:

- **Attributes:** University Name, University Type, Degree Length, In-state Tuition Fees, Out-of-state Tuition Fees.
- **Assumptions:** This entity provides information on tuition fees for different degree lengths and residency status (in-state vs. out-of-state).
- **Why Modeled as an Entity:** Tuition fees information are complex and vary by state, university, and degree length. Modeling this data as an entity allows easy updates, detailed queries, and integration with other financial planning features.
- **Relationships:**
    - **One-to-One with University:** Each university offers a tuition fee structure based on degree length and residency status.

7. User_UnivShortlist Relation:

- **Assumptions:** This relation records the many-to-many relationship between users and universities, indicating which universities a user is interested in.
- **Why Modeled as a relation:** The many-to-many relationship between users and universities needs to be stored separately. Users can be associated with multiple universities, and each university can be linked to multiple users.
- **Relationships:**
    - **Many-to-One with User:** A user can be associated with many universities**.**
    - **Many-to-One with University:** A university can have many users linked to it.

Cardinality of Relationships:

- **User - University (via User_Univ) (Many-to-Many)**: Each user may be interested in multiple universities, and each university can have multiple users.
- **University - Diversity (One-to-One)**: Each university has one diversity profile.
- **University - Tuition Fees (One-to-One)**: Each university offers a unique tuition fee structure (e.g., in-state, out-of-state) based on degree length.
- **University - State-wise Expenses (One-to-One)**: Each university is linked to one state, inheriting its living costs.
- **University - Accommodation (One-to-One)**: Each university has accommodation which includes room and board costs depending on the degree length.

3. Normalize your database. Apply BCNF or 3NF to your schema or show that your schema adheres to one of these normal forms.

- We are using BCNF to ensure that our schema adheres to the highest normal form for eliminating redundancy and dependency anomalies.

Functional Dependencies in Each Entity

1. **User Entity:**

- **Attributes:** Id, FirstName, LastName, EmailId, Password, TuitionFeeBudget, AccommodationBudget.
- **Functional Dependencies:**
  - Id → FirstName, LastName, EmailId, Password, TuitionFeeBudget, AccommodationBudget.
  - **BCNF Check:**The primary key Id uniquely determines all other attributes, making it a superkey. Thus, it adheres to BCNF.

Calculate the minimal basis and then check the schema for BCNF.

Step 1: Minimal Basis

- For the User entity, we have the following attributes and functional dependency:
- Attributes: Id, First Name, Last Name, Email Id, Password, Tuition Fee Budget, Accommodation Budget.
- Functional Dependency:

        - Id → FirstName, LastName, EmailId, Password, TuitionFeeBudget, AccommodationBudget.
    - Steps to calculate the minimal basis:
        - Make all right-hand sides single attributes:
            - The given FD is already in the proper form where the right-hand side is split into individual attributes:
              Id → FirstName
              Id → LastName
              Id → EmailId
              Id → Password
              Id → TuitionFeeBudget
              Id → AccommodationBudget
        - Remove extraneous attributes from the left-hand side:
            - The left-hand side of all dependencies is Id, which is a candidate key, so no attributes can be removed.
        - Remove redundant dependencies:
            - In this case, there are no redundant dependencies as each attribute depends directly on Id.
        - Thus, the minimal basis for the User entity is:
              Id → FirstName
              Id → LastName
              Id → EmailId
              Id → Password
              Id → TuitionFeeBudget
              Id → AccommodationBudget

`

Step 2: BCNF Decomposition

BCNF Criteria:

- A relation is in BCNF if, for every non-trivial functional dependency X → Y, X must be a superkey.
- For the User entity:
    - The only candidate key is Id, and Id is on the left-hand side of all functional dependencies.
    - Since Id is a superkey and is determining all other attributes, the User entity is already in BCNF.

2. University Entity:

- **Attributes:** UniversityName, State.
- **Functional Dependencies:**
    - UniversityName → State
- **BCNF Check:**

- The primary key UniversityName determines the State, confirming that it is a superkey. Therefore, this entity is in BCNF.

Step 1: Minimal Basis (Canonical Cover)

- Make all right-hand sides single attributes:
  - The given functional dependency is already in its simplest form:
  - UniversityName → State
- Remove extraneous attributes from the left-hand side:
  - The left-hand side of the functional dependency is just UniversityName, which is a candidate key for this relation. No extraneous attributes are present, so nothing needs to be removed.
- Remove redundant dependencies:
  - Since there is only one functional dependency, there are no redundant dependencies to remove.
- Thus, the minimal basis for the University entity is:
  - UniversityName → State

Step 2: BCNF Decomposition

- BCNF Criteria:
  - A relation is in BCNF if, for every non-trivial functional dependency X → Y, X must be a superkey.
  - Candidate Key: UniversityName is the candidate key, as it uniquely identifies each tuple (university) in the relation.
  - The functional dependency UniversityName → State has UniversityName as the left-hand side, which is a superkey.
- Since UniversityName is a superkey and it determines State, the University entity satisfies BCNF.

3. Diversity Entity:

- **Attributes:** UniversityName, TotalEnrollment, State, RaceCategory, RaceWiseEnrollment.
- **Functional Dependencies:**
  - UniversityName → TotalEnrollment, State.
  - UniversityName, RaceCategory → RaceWiseEnrollment.
- **BCNF Check:**
  - The primary key (UniversityName) uniquely determines TotalEnrollment and State, while UniversityName and RaceCategory determine RaceWiseEnrollment. In both cases, the determinants are superkeys. Thus, this entity adheres to BCNF.

Step 1: Minimal Basis

- Make all right-hand sides single attributes:
    - We need to break the multi-attribute right-hand side into individual dependencies:
        - From UniversityName → TotalEnrollment, State, we get:
        - UniversityName → TotalEnrollment
        - UniversityName → State
    - The second functional dependency UniversityName, RaceCategory → RaceWiseEnrollment is already in its simplest form.
    - So, the dependencies are now:
        - UniversityName → TotalEnrollment
        - UniversityName → State
        - UniversityName, RaceCategory → RaceWiseEnrollment
- Remove extraneous attributes from the left-hand side:
    - For UniversityName → TotalEnrollment and UniversityName → State, UniversityName is a candidate key for these dependencies, so no extraneous attributes are present.
    - For UniversityName, RaceCategory → RaceWiseEnrollment, both UniversityName and RaceCategory are required to determine RaceWiseEnrollment, so no attributes are extraneous here either.

- Remove redundant dependencies:
    - There are no redundant dependencies, as each functional dependency provides unique information.
    - Thus, the minimal basis for the Diversity entity is:
        - UniversityName → TotalEnrollment
        - UniversityName → State
        - UniversityName, RaceCategory → RaceWiseEnrollment

Step 2: BCNF Decomposition

- BCNF Criteria:
    - A relation is in BCNF if, for every non-trivial functional dependency X → Y, X must be a superkey.
    - Let's evaluate the functional dependencies against this criterion:
        i. UniversityName → TotalEnrollment:
            - Candidate Key: UniversityName uniquely identifies each university, so it is a superkey.
            - BCNF Check: This dependency satisfies BCNF because UniversityName is a superkey.
        ii. UniversityName → State:
            - Candidate Key: UniversityName is still the superkey.

- BCNF Check: This dependency also satisfies BCNF because UniversityName is a superkey.

iii. UniversityName, RaceCategory → RaceWiseEnrollment:

- Candidate Key: The composite key UniversityName, RaceCategory uniquely identifies race-wise enrollment, so it is a superkey.
- BCNF Check: This dependency satisfies BCNF because UniversityName, RaceCategory is a superkey.

- The Diversity entity satisfies BCNF, so no further decomposition is required.

4. StateWiseExpense Entity:

- **Attributes:** State, HousingCost, FoodCost, TransportationCost, HealthcareCost.
- **Functional Dependencies:**
  - State → HousingCost, FoodCost, TransportationCost, HealthcareCost.
- **BCNF Check:**
  - The primary key State determines all other attributes, making it a superkey. Therefore, this entity is in BCNF.

Step 1: Minimal Basis (Canonical Cover)

- Make all right-hand sides single attributes:
  - We need to break the multi-attribute right-hand side into individual dependencies:
  - From State → HousingCost, FoodCost, TransportationCost, HealthcareCost, we get:
    - State → HousingCost
    - State → FoodCost
    - State → TransportationCost
    - State → HealthcareCost
  - Thus, the dependencies are now:
    - State → HousingCost
    - State → FoodCost
    - State → TransportationCost
    - State → HealthcareCost
- Remove extraneous attributes from the left-hand side:
  - In each of these dependencies, State is the only attribute on the left-hand side, and it is necessary for determining the costs, so there are no extraneous attributes.
- Remove redundant dependencies:
  - None of the dependencies are redundant, as each one gives unique information about different costs associated with a state.

- Thus, the minimal basis for the StateWiseExpense entity is:
  - State → HousingCost
  - State → FoodCost
  - State → TransportationCost
  - State → HealthcareCost

Step 2: BCNF Decomposition

- BCNF Criteria:
  - A relation is in BCNF if, for every non-trivial functional dependency X → Y, X must be a superkey.
  - Candidate Key: State uniquely identifies each tuple in this relation, so it is the superkey.
  - The functional dependencies State → HousingCost, State → FoodCost, State → TransportationCost, and State → HealthcareCost all have State as the left-hand side, which is a superkey.
- Since State is a superkey and it determines all the other attributes, the relation satisfies BCNF.

5. Accommodation Entity:

- **Attributes:** UserId, UniversityName, DegreeLength, RoomAndBoardCost.
- **Functional Dependencies:**
  - UserId, UniversityName, DegreeLength → RoomAndBoardCost.
- **BCNF Check:**
  - The combination of UserId, UniversityName, and DegreeLength is the primary key, and it uniquely determines RoomAndBoardCost. As such, this adheres to BCNF.

Step 1: Minimal Basis

- Make all right-hand sides single attributes:
  - The given functional dependency is already in its simplest form:
    - UserId, UniversityName, DegreeLength → RoomAndBoardCost

- Remove extraneous attributes from the left-hand side:
  - We need to check if any of the attributes in the left-hand side (UserId, UniversityName, DegreeLength) are extraneous. To do this, we need to see if the functional dependency can hold without any one of these attributes:
    - Remove UserId: If we remove UserId, we would have UniversityName, DegreeLength → RoomAndBoardCost. This would imply that room and board cost depends only on the university and the degree length, regardless of the user. However,

room and board costs may vary for different users even at the same university for the same degree length (e.g., based on different room preferences). Therefore, UserId cannot be removed.
        - Remove UniversityName: If we remove UniversityName, we would have UserId, DegreeLength → RoomAndBoardCost. This would imply that the room and board cost depends on the user and degree length, without considering the specific university. This is not reasonable because different universities have different accommodation costs. Therefore, UniversityName cannot be removed.
        - Remove DegreeLength: If we remove DegreeLength, we would have UserId, UniversityName → RoomAndBoardCost. This would imply that room and board costs depend only on the user and university, regardless of the length of the degree, which is not accurate since room and board costs can vary by the degree length. Therefore, DegreeLength cannot be removed.
    - Thus, no attributes are extraneous, and the dependency remains as:
        - UserId, UniversityName, DegreeLength → RoomAndBoardCost
- Remove redundant dependencies:
    - There are no redundant dependencies, as we have only one functional dependency.
    - Thus, the minimal basis for the Accommodation entity is:
        - UserId, UniversityName, DegreeLength → RoomAndBoardCost.

Step 2: BCNF Decomposition

- BCNF Criteria:
    - A relation is in BCNF if, for every non-trivial functional dependency X → Y, X must be a superkey.
        - Candidate Key: The composite key UserId, UniversityName, DegreeLength is the candidate key because it uniquely identifies each tuple in the relation.
        - The functional dependency UserId, UniversityName, DegreeLength → RoomAndBoardCost has the composite key UserId, UniversityName, DegreeLength as its left-hand side, which is a superkey.
- Since UserId, UniversityName, DegreeLength is a superkey, this dependency satisfies BCNF.
- The Accommodation entity is already in BCNF, and no further decomposition is required.

6. TuitionFees Entity:

- **Attributes:** UniversityName, UniversityType, StateCode, State, DegreeLength, InStateTuitionFees, OutOfStateTuitionFees.
- **Functional Dependencies:**
    - UniversityName, DegreeLength → InStateTuitionFees, OutOfStateTuitionFees.
    - UniversityName → UniversityType, StateCode, State.
- **BCNF Check:**
    - The combination of UniversityName and DegreeLength forms a candidate key that uniquely determines the tuition fees, while UniversityName alone determines other attributes. Since all determinants are superkeys, this entity adheres to BCNF.

Step 1: Minimal Basis

- Make all right-hand sides single attributes:
    - We need to break the multi-attribute right-hand sides into individual dependencies:
        - From UniversityName, DegreeLength → InStateTuitionFees, OutOfStateTuitionFees, we get:
            - UniversityName, DegreeLength → InStateTuitionFees
            - UniversityName, DegreeLength → OutOfStateTuitionFees
        - From UniversityName → UniversityType, StateCode, State, we get:
            - UniversityName → UniversityType
            - UniversityName → StateCode
            - UniversityName → State
    - Thus, the dependencies are now:
        - UniversityName, DegreeLength → InStateTuitionFees
        - UniversityName, DegreeLength → OutOfStateTuitionFees
        - UniversityName → UniversityType
        - UniversityName → StateCode
        - UniversityName → State
- Remove extraneous attributes from the left-hand side:
    - Now, we need to check if any attributes on the left-hand side are extraneous:
        - For UniversityName, DegreeLength → InStateTuitionFees and UniversityName, DegreeLength → OutOfStateTuitionFees:
            - Both UniversityName and DegreeLength are needed to uniquely determine the tuition fees, so no attributes can be removed.
        - For UniversityName → UniversityType, UniversityName → StateCode, and UniversityName → State:
            - UniversityName is the only attribute on the left-hand side, and it's necessary, so no attributes can be removed here either.
- Remove redundant dependencies:

    - None of the dependencies are redundant, as each provides unique information.
    - Thus, the minimal basis for the TuitionFees entity is:
        - UniversityName, DegreeLength → InStateTuitionFees
        - UniversityName, DegreeLength → OutOfStateTuitionFees
        - UniversityName → UniversityType
        - UniversityName → StateCode
        - UniversityName → State

Step 2: BCNF Decomposition

- BCNF Criteria:
    - A relation is in BCNF if, for every non-trivial functional dependency X → Y, X must be a superkey.
        - Candidate Key:
            - From the functional dependencies, we can see that UniversityName, DegreeLength can uniquely identify the InStateTuitionFees and OutOfStateTuitionFees.
            - However, UniversityName alone can determine UniversityType, StateCode, and State.
            - Therefore, the candidate key is UniversityName, DegreeLength for the whole relation.
- Analyze the Functional Dependencies:
    - For UniversityName, DegreeLength → InStateTuitionFees and UniversityName, DegreeLength → OutOfStateTuitionFees:
        - UniversityName, DegreeLength is a superkey, so these dependencies are in BCNF.
    - For UniversityName → UniversityType, UniversityName → StateCode, and UniversityName → State:
        - Here, UniversityName is not a superkey, as it does not uniquely identify all attributes in the relation (it doesn't determine DegreeLength, InStateTuitionFees, and OutOfStateTuitionFees).
        - These dependencies violate BCNF because UniversityName is not a superkey.

Step 3: BCNF Decomposition

- Since UniversityName → UniversityType, StateCode, State violates BCNF, we decompose the relation.
- We split the relation into two smaller relations:
    - Relation 1:
        - Attributes: UniversityName, UniversityType, StateCode, State
        - Functional dependencies: UniversityName → UniversityType, UniversityName → StateCode, UniversityName → State

        - In this relation, UniversityName is the candidate key, and the relation is in BCNF.
    - Relation 2:
        - Attributes: UniversityName, DegreeLength, InStateTuitionFees, OutOfStateTuitionFees
        - Functional dependencies:
            - UniversityName, DegreeLength → InStateTuitionFees
            - UniversityName, DegreeLength → OutOfStateTuitionFees
        - In this relation, UniversityName, DegreeLength is the candidate key, and the relation is in BCNF.

Conclusion:

- After decomposition, the TuitionFees entity is split into two relations in BCNF:
    - Relation 1:
        - Attributes: UniversityName, UniversityType, StateCode, State
        - Functional dependencies:
            - UniversityName → UniversityType
            - UniversityName → StateCode
            - UniversityName → State
    - Relation 2:
        - Attributes: UniversityName, DegreeLength, InStateTuitionFees, OutOfStateTuitionFees
        - Functional dependencies:
            - UniversityName, DegreeLength → InStateTuitionFees
            - UniversityName, DegreeLength → OutOfStateTuitionFees
- This decomposition satisfies BCNF.

Conclusion:

All entities in the schema satisfy the conditions for BCNF.

a. Each non-trivial functional dependency has a determinant that is a superkey.
b. There are no partial dependencies or transitive dependencies among the non-prime attributes.

4. Convert your conceptual database design (ER/UML) to the logical design (relational schema). Note that a relational schema is NOT an SQL DDL command.

Relational Schema

1. **User:** User (Id: INT [PK], FirstName: VARCHAR(255), LastName: VARCHAR(255), EmailId: VARCHAR(255), Password: VARCHAR(255), TuitionFeeBudget: INT, AccommodationBudget: INT).

2. **University:** University (UniversityName: VARCHAR(255) [PK], State: VARCHAR(100)).

3. **Diversity:** Diversity (UniversityName: VARCHAR(255) [PK, FK to University.UniversityName], TotalEnrollment: INT, State: VARCHAR(100), RaceCategory: VARCHAR(100), RaceWiseEnrollment: INT).

4. **StateWiseExpense:** StateWiseExpense (State: VARCHAR(100) [PK], HousingCost: INT, FoodCost: INT, TransportationCost: INT, HealthcareCost: INT).

5. **Accommodation:** Accommodation (UserId: INT [FK to User.Id], UniversityName: VARCHAR(255) [FK to University.UniversityName], DegreeLength: INT, RoomAndBoardCost: INT, PRIMARY KEY(UserId, UniversityName, DegreeLength)).

6. **TuitionFees:** TuitionFees (UniversityName: VARCHAR(255) [PK, FK to University.UniversityName], UniversityType: VARCHAR(100), StateCode: VARCHAR(10), State: VARCHAR(100), DegreeLength: INT, InStateTuitionFees: INT, OutOfStateTuitionFees: INT).

7. **User_UnivShortlist** : User_UnivShortlist(UserId: INT [FK to User.Id], UniversityName: VARCHAR(255) [FK to University.UniversityName], PRIMARY KEY(UserId, UniversityName))