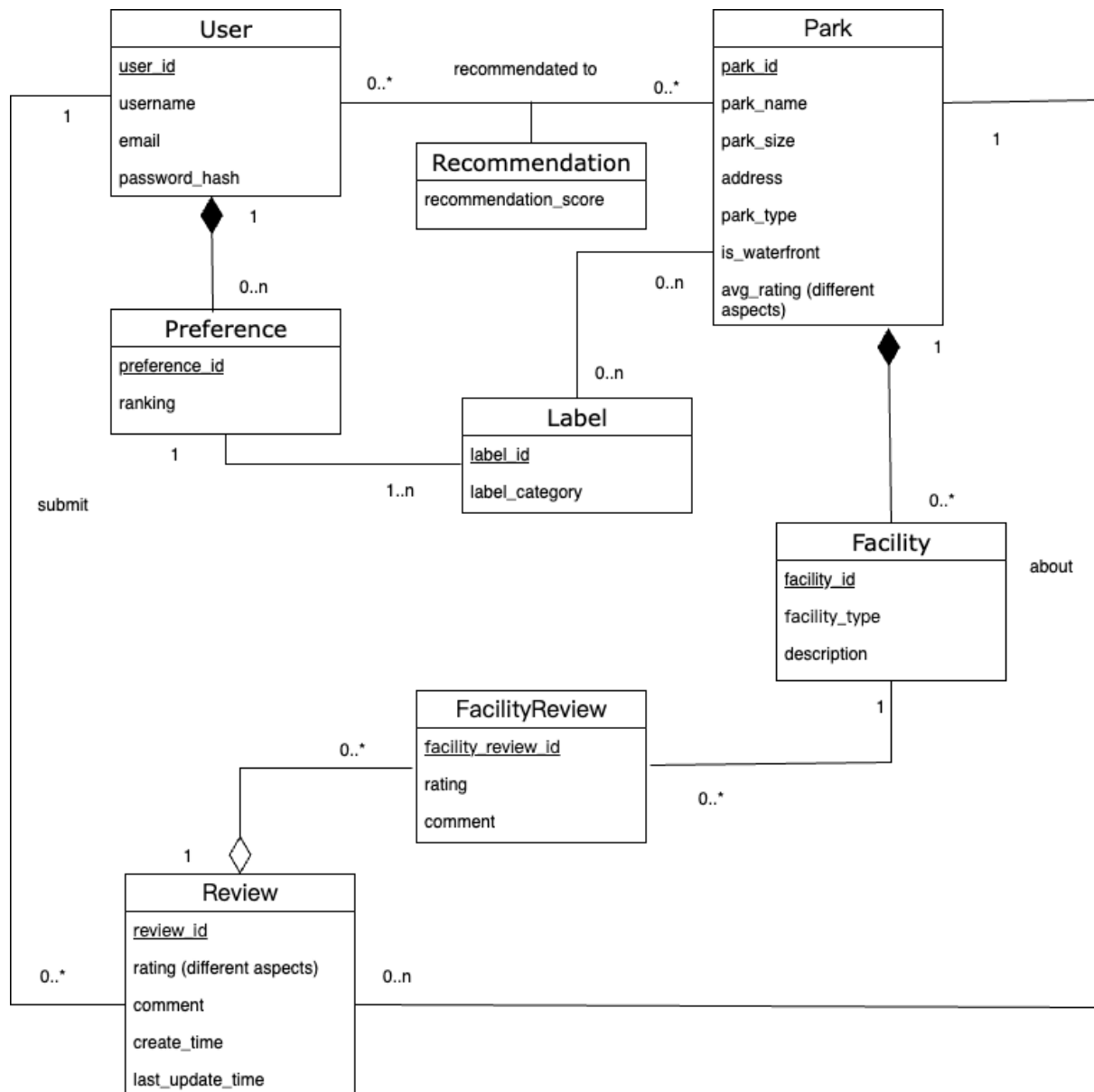


# PT1 - (Stage 2)

Team020 IShowCode

## 1.UML Diagram



## 2. Entity Descriptions and Assumptions

### User

**Descriptions:**

Represents an individual user who interacts with the whole system

**Assumptions:**

- Each user has a unique user\_id and a unique email address.
- A user can submit multiple reviews, preferences and recommendations.

**Why build an entity:**

- Each user interacts with multiple parts of the system, like review, preference, recommendations...

### Preference

**Descriptions:**

Marks a user's interests used to personalize park recommendations.

**Assumptions:**

- A user can have multiple preferences to express different interests.
- Each preference has a unique preference\_id.
- ranking is the weight which present the important level for users.

**Why build an entity:**

- Preferences represent user interests and link to label.

### Label

**Descriptions:**

Represents a classification type for park, like friendly to runner.

**Assumptions:**

- Each label has a unique label\_id and a related label\_name.
- Labels can be related for both user's preference and park's attributes.

**Why build an entity:**

- Labels are the bridge between park's attributes and user's preference.

### Recommendation

**Descriptions:**

Show the recommendation results generated by the system for users.

**Assumptions:**

- Each recommendation includes a recommendation score (recommendation\_score) calculated by the system.
- Recommendations are generated based on user preferences, label weights, and historical reviews.

**Why build an entity:**

- Recommendation is the score generated by our recommendation system linked with users and parks.

## Park

### **Descriptions:**

Represents a park with its attributes, information and user review.

### **Assumptions:**

- Each park has a unique park\_id, and must include a name and an address.
- is\_waterfront is a boolean attribute indicating whether a park adjacent to a lake or pool.
- A park can include multiple facilities and receive reviews for multiple users.

### **Why build an entity:**

- Park stores its own attributes like size, location, reviews...

## Facility

### **Descriptions:**

Record a facility located within a specific park.

### **Assumptions:**

- Each facility has a unique facility\_id and belongs to exactly one park.
- A park can have multiple facilities or none at all.
- Facility information includes type and description.

### **Why build an entity:**

- Facilities have distinct information and can receive separate reviews.

## FacilityReview

### **Descriptions:**

Record the reviews by user for a specific facility.

### **Assumptions:**

- Each facility review has a unique facility\_review\_id and is submitted by a user for a specific facility.
- A user can submit multiple reviews for the same facility.

### **Why build an entity:**

- Facility reviews contain ratings by user and comments about facilities.

## Review

### **Descriptions:**

Represents user's review and rating of a specific park.

### **Assumptions:**

- Each review has a unique review\_id and is submitted by users to park.
- Each review includes a rating and creation time.
- Users can submit multiple reviews for a park.

### **Why build an entity:**

- Reviews record user interactions with parks and have their own attributes.

### 3. Cardinality

Relationship	Cardinality	Description
User – Review	(1, 0..*)	One user can submit zero to multiple reviews
Park – Review	(1, 0..*)	One park can have zero to multiple reviews. Each review only reviews on one park
User – Park (Recommendation)	(0..*, 0..*)	The recommendation system can recommend multiple parks to multiple users
User – Preference (Composition)	(1, 0..*)	One user can have zero to multiple park preferences with different rankings to distinguish the level of preference. When a user is deleted, his/her preferences should be deleted as well.
Park – Facility (Composition)	(1, 0..*)	One park can have zero to multiple facilities. When a park is deleted, its facilities should also be deleted.
Facility – FacilityReview	(1, 0..*)	One facility can have zero to multiple facility reviews
Preference – Label	(1, 1..*)	One preference can have one to multiple labels, which are correlated to the relationship between labels and parks. The recommendation uses overlapping labels in Preference-Label and Label-Parks to find the best park recommendations for the user
Park – Label	(0..*, 0..*)	Zero to multiple parks can have zero to multiple labels.
Review – FacilityReview (Aggregation)	(1, 0..*)	One review owns zero to multiple facility reviews, as

		there may be multiple facilities in the park. When a review is deleted, the Facility Review may remain in the database.
--	--	---

## 4 Normalize your database

### 1. Normalization Goal

The database for FindMyPark NYC is normalized to at least Third Normal Form (3NF), and most tables satisfy Boyce-Codd Normal Form (BCNF).

Normalization ensures data consistency, removes redundancy, and eliminates update anomalies.

### 2. Functional Dependencies and Keys

Each table's primary key (PK) and main functional dependencies (FDs) are defined as follows:

User(user\_id [PK], username, email, password\_hash)

FDs: user\_id → username, email, password\_hash

Park(park\_id [PK], park\_name, park\_size, address, park\_type, is\_waterfront)

FDs: park\_id → park\_name, park\_size, address, park\_type, is\_waterfront

Facility(facility\_id [PK], park\_id [FK], facility\_type, description)

FDs: facility\_id → park\_id, facility\_type, description

Review(review\_id [PK], user\_id [FK], park\_id [FK], comment, create\_time, last\_update\_time)

FDs: review\_id → user\_id, park\_id, comment, create\_time, last\_update\_time

FacilityReview(facility\_review\_id [PK], review\_id [FK], facility\_id [FK], comment)

FDs: facility\_review\_id → review\_id, facility\_id, comment

Preference(preference\_id [PK], user\_id [FK], created\_at)

FDs: preference\_id → user\_id, rank

Label(label\_id [PK], label\_name)

FDs: label\_id → label\_name

Recommendation(user\_id [FK], park\_id [FK], generated\_at, recommendation\_score,  
PK(user\_id, park\_id, generated\_at))

FDs: (user\_id, park\_id, generated\_at) → recommendation\_score

### 3. Detected Violations and Decompositions

- a. Park.avg\_rating (different aspects)
  - Problem: Multiple aspect ratings in one field violate 1NF.
  - Fix: Remove this field; compute averages from `ReviewRating` dynamically.
- b. Review.rating (different aspects)
  - Problem: Same issue—non-atomic values.
  - Fix: Move aspect-specific ratings into a new table `ReviewRating`.
- c. Park / Label / Preference relationship
  - Problem: The UML shows a three-way relationship between preferences, labels, and parks; if stored separately, it creates partial dependencies.
  - Fix: Combine into `PreferenceLabelWeight(preference\_id, label\_id, label\_weight)` with a composite primary key.
- d. Address atomicity (optional improvement)
  - Fix: Split into street, city, state, zipcode if detailed queries are required.

### 4. Normal Form Proof

- 1NF: All attributes are atomic (no repeating groups).
- 2NF: All non-key attributes depend on the whole primary key (no partial dependencies).
- 3NF: There are no transitive dependencies; every non-key attribute depends only on the key.
- BCNF: For most tables, every determinant is a candidate key.

## 5. Final Relational Schema

User

```
(  
    user_id: INT [PK],  
    username: VARCHAR(50),  
    email: VARCHAR(255),  
    password_hash: VARCHAR(255)  
)
```

Park

```
(  
    park_id: INT [PK],  
    park_name: VARCHAR(200),  
    park_size: DECIMAL(10,2),  
    address: VARCHAR(300),  
    park_type: VARCHAR(50),  
    is_waterfront: BOOLEAN  
)
```

avg\_rating: INT  
)

#### Facility

(  
    facility\_id: INT [PK],  
    park\_id: INT [FK to Park.park\_id],  
    facility\_type: VARCHAR(80),  
    description: VARCHAR(300)  
)

#### Review

(  
    review\_id: INT [PK],  
    rating\_value: DECIMAL(2,1),  
    user\_id: INT [FK to User.user\_id],  
    park\_id: INT [FK to Park.park\_id],  
    comment: VARCHAR(1000),  
    create\_time: TIMESTAMP,  
    last\_update\_time: TIMESTAMP  
)

#### FacilityReview

(  
    facility\_review\_id: INT [PK],  
    review\_id: INT [FK to Review.review\_id],  
    facility\_id: INT [FK to Facility.facility\_id],  
    comment: VARCHAR(600)  
)

#### Preference

(  
    preference\_id: INT [PK],  
    user\_id: INT [FK to User.user\_id],  
    ranking: INT  
)

#### Label

(  
    label\_id: INT [PK],  
    label\_name: VARCHAR(50)  
)

#### Recommendation

(  
    user\_id: INT [FK to User.user\_id],  
    park\_id: INT [FK to Park.park\_id],

```
generated_at: TIMESTAMP,  
recommendation_score: DECIMAL(6,3),  
[PK(user_id, park_id, generated_at)]  
)
```