Rate ‘Em

Senior Project - Conceptual Database Design

Keben Carrillo, Elena Castaneda, Carlos Hernandez, Laura Moreno

Tables of Contents

Conceptual Database Design

1.1 - Entity Type Descriptions

Entity name: User

Description: This provides all the basic information about a user shared between the two user types: Tenant and Landlord.

Relationship(s): User HAS Comment. User LEAVES Comment. User RATES Comment.

Candidate Keys: Email, PhoneNumber

Primary Key: Email

Strong or Weak Entity: Strong

Attribute Description:

Name	Description	Domain /Type	Value-Range	Default Value	Null Value Allowed ? (Y/N)	Unique Attribute ?	Single or Multi Value?	Simple or Derived or Composite ?
Email	User's email address	Varchar	A-Z, a-z, 0-9	None	No	Yes	Single	Simple
Password	User's account password	Varchar	A-Z, a-z, 0-9	None	No	No	Single	Simple
Name	User's full name	String	A-Z	None	No	No	Single	Composite
PhoneNumber	User's phone number	Int	0-9	None	No	Yes	Single	Simple
Rating	The User's rating left by other users	Int	0-max	None	Yes	No	Single	Derived

Entity name: Comment

Description: Comments are the messages left by users on a user's account for the public to view.

Relationship(s): Comment is LEFT (LEAVES) by a User. Comment is RATED (RATES) by a User. Comment is FOR (HAS) a User.

Candidate Keys: CommentID

Primary Key: CommentID

Strong or Weak Entity: Weak

Attribute Description:

Name	Description	Domain/ Type	Value-Range	Default Value	Null Value Allowed? (Y/N)	Unique Attribute?	Single or Multi Value?	Simple or Derived or Composite?
CommentID	Comment's identification key	Serial	0-9	None	No	Yes	Single	Simple
Message	The comment's message	Longtext	A-Z, a-z, 0-9	None	No	No	Single	Simple
Date	Comment's post date	Datetime	0-9999	Current date	No	No	Single	Simple
Dislikes	The number of dislikes on a comment post.	Int	0-max	0	No	No	Single	Composite
Likes	The number of likes on a comment post.	Int	0-max	0	No	No	Single	Composite

Entity name: Property

Description: The basic information for the property owned by Landlord users.

Relationship(s): Property OWNED (OWNS) by Landlord.

Candidate Keys: PropertyID

Primary Key: PropertyID

Strong or Weak Entity: Weak

Attribute Description:

Name	Description	Domain /Type	Value-Range	Default Value	Null Value Allowed ? (Y/N)	Unique Attribute ?	Single or Multi Value?	Simple or Derived or Composite ?
PropertyID	Property's identification key	Serial	0-9	None	No	Yes	Single	Simple
Type	Type of Property	String	A-Z, a-z	None	No	No	Multival.	Simple
NumberofRooms	How many bedrooms are there	Int	1-max	None	No	No	Single	Simple
NumOfBathrooms	How many bathrooms are there	Int	1-max	None	No	No	Single	Simple
Price	The renting price	Double	min-max	None	No	No	Single	Simple
Address	Contains state, city, and zipcode of property	String	A-Z,a-z, 0-9	None	No	No	Single	Composite

1.2 - Relationship Type Description

Name	Description	Entities	Cardinality	Participation	Attributes
LEAVES	**Users can leave comments on other user accounts.**	**User and Comment**	**(1,M)**	**User (Parital) Comment (Total)**	**None**
HAS	**A user account can have**	**User and Comment**	**(1,M)**	**User (Partial)**	**None**

	comments that were left by users.			**Comment (Partial)**	
CommentRates	**A user can rate comments.**	**User and Comment**	**(M,N)**	**User (Partial) Comment (Partial)**	**Rating** - The rating for comments, such as likes and dislikes. - Int - 0-max - 0 - No null value allowed - Not unique -Single -Simple
UserRates	**Users such as Tenants and Landlords can rate each other.**	**Landlord and Tenant**	**(M,N)**	**Landlord (Partial) Tenant (Partial)**	**Stars** - The star rating for the individual users. - Int - 0-max - No default value - Yes null value allowed - Not unique - Single - Simple
OWNS	**A landlord owns a property that they offer people to rent, so they share the basic information.**	**Landlord and Property**	**(1,M)**	**Landlord (Total) Property (Total)**	**None**
OCCUPIES	**A tenant has rented or is currently renting a property.**	**Tenant and Property**	**(M,M)**	**Tenant (Partial) Property (Partial)**	**Start** - The start date of the lease. - DATETIME - 0-Current Date - Current Date - No null value allowed - Unique - Single - Simple **End** - The end date of the lease. - DATETIME - Greater than the ‘Start’ date

					- No default value - Yes null value allowed - Not unique - Single - Simple **Stars** - The star rating for the properties that a tenant has stayed at. - Int - 0-max - No default value - Yes null value allowed - Not unique - Single - Simple

Specialization/Generalization Entities:

- Generalization Entity: User
- Specialization Entities: Tenant and Landlord
- Participation constraint: Total Participation
  - User (Total) and Tenant (Total).
  - User (Total) and Landlord (Total).
- A disjoint constraint.

Union Types: None

Entity name: Tenant

Description: Users can register as a Tenant.

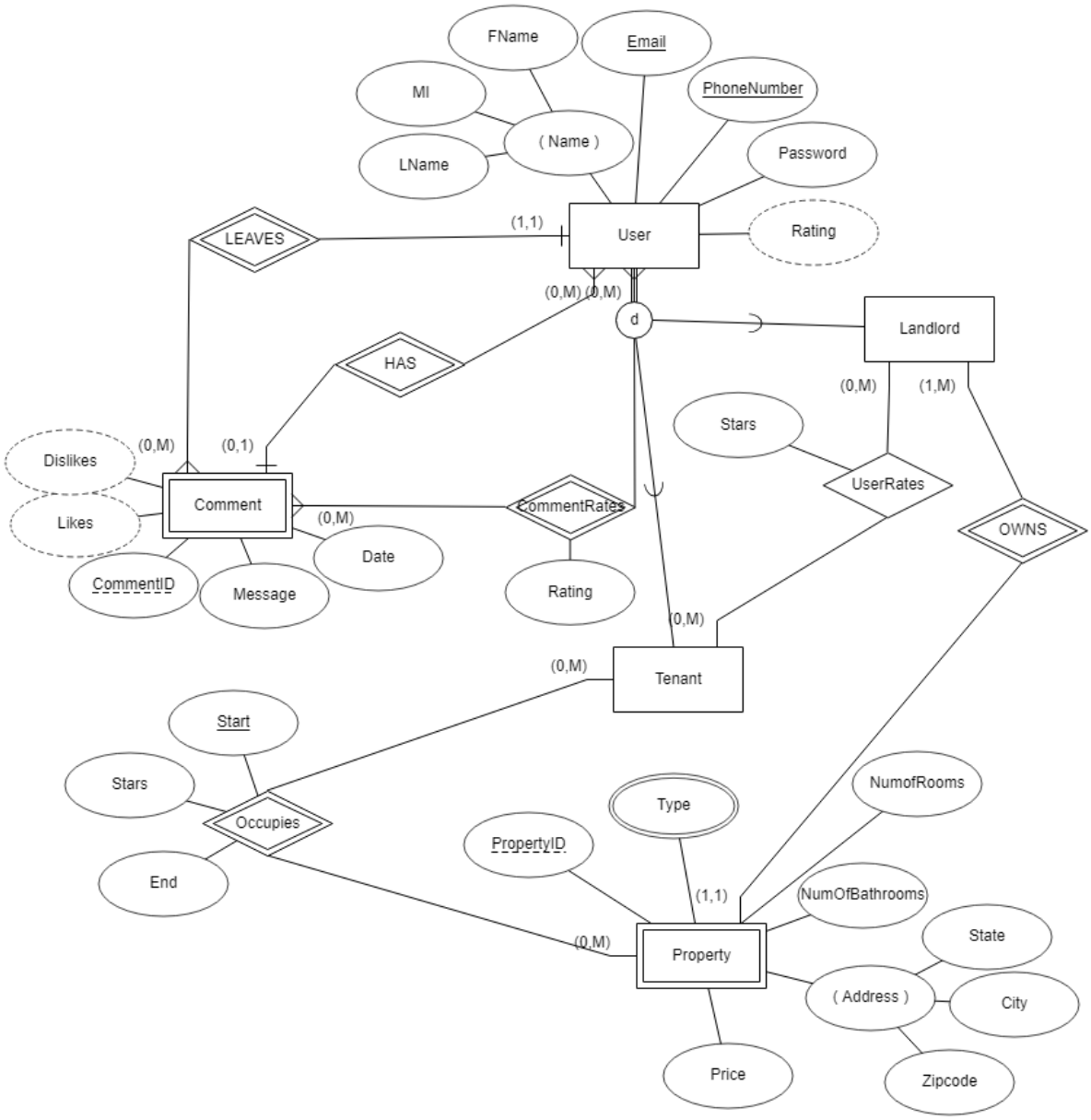
Relationship(s): Tenant RATES Landlord.

Entity name: Landlord

Description: Users can register as a Landlord.

Relationship(s): Landlord RATES Tenant. Landlord OWNS Property.

1.3 - ER Diagram



The ER and Relational Models

2.1 - Relation Schema

Strong Entity Relations:

User

Email	PhoneNumber	FName	MI	LName	Password

Primary Key(s): Email **Foreign Key(s)**: N/A.

Candidate Key(s): Email, PhoneNumber **Derived Attribute(s):** UserRating

Semantic Constraints:

Email - Domain: Varchar(50). Cannot be NULL

PhoneNumber - Domain: Varchar(13). Cannot be NULL. Unique

FName - Domain: Text. Cannot be NULL

MI - Domain: Text. Not required, can be NULL

LName - Domain: Text. Cannot be NULL

Password - Domain: Text. Must be 6 characters long. Case sensitive. Cannot be NULL

Tenant

Email

Primary Key(s): Email

Foreign Key(s): Email to Email in User relation

Candidate Key(s): Email **Derived Attribute(s):** UserRating

Semantic Constraints:

Email - Domain: Varchar(50). Cannot be NULL

Landlord

Email

Primary Key(s): Email

Foreign Key(s): Email to Email in User relation

Candidate Key(s): Email **Derived Attribute(s):** UserRating

Semantic Constraints:

Email - Domain: Varchar(50). Cannot be NULL

Weak Entity Relations:

Property

LEmail	PropertyID	NumOfRooms	NumOfBathrooms	Price	State	City	Zipcode

Primary Key(s): LEmail, PropertyID

Foreign Key(s): LEmail to Email in Landlord relation

Candidate Key(s): LEmail, PropertyID

Semantic Constraints:

LEmail - Domain: Varchar(50). Cannot be NULL

PropertyID - Domain: Serial. Cannot be NULL

NumOfRooms - Domain: Int. Cannot be NULL

NumOfBathrooms - Domain: Int. Cannot be NULL

Price - Domain: Decimal(10,2). Cannot be NULL.

State - Domain: Tinytext. Cannot be NULL.

City - Domain: TinyText. Cannot be NULL..

Zipcode - Domain: Int(5). Cannot be NULL.

Comment

UEmail	CommentID	ForEmail	Message	Date

Primary Key(s): CommentID

Foreign Key(s): UEmail and ForEmail to Email User relation

Candidate Key(s): CommentID **Derived Attribute(s):** Dislikes, Likes

Semantic Constraints:

UEmail - Domain: Varchar(50). Cannot be NULL

CommentID - Domain: Serial. Cannot be NULL

Message - Domain: Longtext. Cannot be NULL

Date - Domain: Datetime. Cannot be NULL

Binary M:N Relations:

CommentRates

UEmail	CommentID	Rating

Primary Key(s): UEmail, CommentID **Foreign Key(s)**: UEmail, CommentID

Candidate Key(s): UEmail, CommentID

Semantic Constraints:

UEmail - Domain: Varchar(50). Cannot be NULL

CommentID - Domain: Serial. Cannot be NULL

Rating - Domain: Int. Cannot be NULL.

UserRates

UEmail	ForEmail	Stars

Primary Key(s): UEmail, ForEmail

Foreign Key(s): UEmail, ForEmail to User relation

Candidate Key(s): UEmail, ForEmail

Semantic Constraints:

If UEmail is a tenant then ForEmail is a landlord. Vice versa.

UEmail - Domain: Varchar(50). Cannot be NULL

Stars - Domain: Smallint [1-5]. Cannot be NULL.

1:M Relations:

Occupies

TEmail	PropertyID	Start	End	Stars

Primary Key(s): TEmail, PropertyID, Start

Foreign Key(s): PropertyID to Property relation and TEmail to Email User relation

TEmail - Domain: Varchar(50). Cannot be NULL

PropertyID - Domain: Serial. Cannot be NULL

Start - Domain: Datetime. Cannot be NULL

End - Domain: Datetime. Can be NULL

Stars - Domain: Smallint [1-5]. Cannot be NULL.

Multi-Valued Attributes Relations:

PropertyType

PropertyID	Type

Primary Key(s): PropertyID

Foreign Key(s): PropertyID, PropertyID to Property relation

PropertyID - Domain: Serial. Cannot be NULL

Type - Domain: String. Cannot be NULL. Property Type.

2.2 - Design Of Queries

Relations:
User(Email, PhoneNumber, FName, MI, LName, Password)
Tenant(Email)
Landlord(Email)
Property(PropertyID, LEmail, NumOfRooms, NumOfBathrooms, Price, State, City, Zipcode)
Occupies(TEmail, PropertyID, Start, End, Stars)
Comment(CommentID, UEmail, ForEmail, Message, Date)
CommentRates(UEmail, CommentID, Rating)
UserRates(UEmail, ForEmail, Stars)
PropertyType(PropertyID, Type)

Queries:

Queries targeting one relation (no joins necessary) with some condition based on the relation's attribute

1. Select the tenant named “Homer Simpson”.

Queries targeting 2 relations (1 join between them)

2. Select the first name of the user who left a comment.

Queries targeting 3 relations (2 joins between them)

3. Select the landlord's name and their property that is in Bakersfield, California and is listed with a renting price under $500.
4. Select the properties that each tenant has rented.

Queries to select a relation with either the largest or smallest attribute value in its set

5. Select the comment with the most number of likes.
6. Select the Landlord with the highest rating.

Queries that use relational division

7. Select the tenant who rates all landlords in Bakersfield, CA.
8. Select landlord’s who own at least one of every property type.

2.3 - Relational Calculus/Algebra Expressions

Relations:

User(Email, PhoneNumber, FName, MI, LName, Password)
Tenant(Email)
Landlord(Email)
Property(PropertyID, LEmail, NumOfRooms, NumOfBathrooms, Price, State, City, Zipcode)
Occupies(TEmail, PropertyID, Start, End, Stars)
Comment(CommentID, UEmail, ForEmail, Message, Date)
CommentRates(UEmail, CommentID, Rating)
UserRates(UEmail, ForEmail, Stars)
PropertyType(PropertyID, Type)

1. Select the user that is a tenant named “Homer Simpson”.

   {t | Tenant(t) ^ ∃u(User(u) ^ u.Email = t.Email ^ u.FName ^ u.LName = “Homer Simpson”) }

   // Both work

   {u | User(u) ^ u.FName = “Homer” ^ u.LName = “Simpson” ^ ∃t(Tenant(t) ^ u.Email = t.Email) }

2. Select the first name of the user who left a comment.

   {u.FName | User(u) ^ ∃d(Comment(d) ^ d.UEmail = u.Email) }

3. Select the landlord's name and their property that is in Bakersfield, California and is listed with a renting price under $500.

   {u.FName, u.MI, u.LName | User(u) ^ ∃l(Landlord(l) ^ l.Email = u.Email ^ ∃p(Property(p) ^ p.LEmail = l.Email ^ p.City = “Bakersfield” ^ p.State = “California” ^ p.Price < 500)) }

4. Select the properties that each tenant has rented.

   {p | Property(p) ^ ∃o(Occupies(o) ^ p.PropertyID = o.PropertyID ^ ∃t(Tenant(t) ^ o.TEmail = t.Email) }

5. Select the comment with the most number of likes.

   R(CommentID, Ratings) ← CommentID $G_{SUM(Rating)}$CommentRates

   S ← $G_{CommentID, MAX(Ratings)}$R

   F ← S * Comment

6. Select the Landlord with the highest rating.

   R(ForEmail, Ratings) ← ForEmail $G_{AVG(Stars)}$UserRates

   S ← $G_{ForEmail, MAX(Ratings)}$R ⋈ Landlord

   F ← π_{Email} (Landlord ⋈ S (Email = ForEmail))

7. Select the tenant who rates all landlords in Bakersfield, CA.

   L ← ρ (LEmail, ForEmail) (π_{LEmail}($\sigma_{State = "California" \wedge City = "Bakersfield"}$ (Property)))

   R ← $\pi_{UEmail, ForEmail}$(UserRates) ÷ L

   F ← Tenant * R

8. Select landlord's who own at least one of every property type.

   R1 ← $\pi_{Type, LEmail}$((PropertyType) ⋈ Property) ÷ π_{Type} (PropertyType)

   F ← Landlord * R1

2.4 - Application to Relational Model

Relation	Normal Form	Anomalies	Change Required
User	3NF	No	No
Tenant	3NF	No	No
Landlord	3NF	No	No
Property	3NF	No	No
Occupies	3NF	No	No
Comment	3NF	No	No
CommentRates	3NF	No	No
UserRates	3NF	No	No
PropertyType	BCNF	No	No

Relations:
User(Email, PhoneNumber, FName, MI, LName, Password)
Tenant(Email)
Landlord(Email)
Property(PropertyID, LEmail, NumOfRooms, NumOfBathrooms, Price, State, City, Zipcode)

Occupies(TEmail, PropertyID, Start, End, Stars)
~~**PropertyRates**(TEmail, PropertyID, Date, Stars)~~
Comment(CommentID, UEmail, ForEmail, Message, Date)
CommentRates(UEmail, CommentID, Rating)
UserRates(UEmail, ForEmail, Stars)
PropertyType(PropertyID, Type)

Query Implementation

Relations:

User(Email, PhoneNumber, FName, MI, LName, Password)
Tenant(Email)
Landlord(Email)
Property(PropertyID, LEmail, NumOfRooms, NumOfBathrooms, Price, State, City, Zipcode)
Occupies(TEmail, PropertyID, Start, End, Stars)
Comment(CommentID, UEmail, ForEmail, Message, Date)
CommentRates(UEmail, CommentID, Rating)
UserRates(UEmail, ForEmail, Stars)
PropertyType(PropertyID, Type)

9. Select the user that is a tenant named “Homer Simpson”.

   SELECT DISTINCT Tenant.* FROM Tenant NATURAL JOIN User WHERE User.FName = "Homer" AND User.LName = "Simpson";

10. Select the first name of the user who left a comment.

    SELECT DISTINCT User.FName FROM User NATURAL JOIN Comment;

11. Select the landlord's name and their property that is in Bakersfield, California and is listed with a renting price under $500.

    SELECT DISTINCT User.FName, User.MI, User.LName FROM User NATURAL JOIN Landlord NATURAL JOIN Property WHERE Property.City = "Bakersfield" AND Property.State = "California" AND Property.Price < 500;

12. Select the properties that each tenant has rented.

    {p | Property(p) ^ ∃ o(Occupies(o) ^ p.PropertyID = o.PropertyID ^ ∃ t(Tenant(t) ^ o.TEmail = t.Email) }

    SELECT DISTINCT Property.* FROM Property NATURAL JOIN Occupies INNER JOIN Tenant ON Occupies.TEmail = Tenant.Email;

13. Select the comment with the most number of likes.

R(CommentID, Ratings) ← CommentID $G_{SUM(Rating)}$CommentRates

S ← $G_{CommentID, MAX(Ratings)}$R

F ← S * Comment

```
SELECT Comment.*
FROM
Comment INNER JOIN
(
        SELECT SUM(Rating) AS TotalRating, CommentID
        FROM CommentRates NATURAL JOIN Comment
        GROUP BY CommentID
) AS Ratings
ON Ratings.CommentID = Comment.CommentID
ORDER BY Ratings.TotalRating DESC
LIMIT 1;
```

14. Select the Landlord with the highest rating.

R(ForEmail, Ratings) ← ForEmail $G_{AVG(Stars)}$UserRates

S ← $G_{ForEmail, MAX(Ratings)}$R ⋈ Landlord

F ← π_{Email} (Landlord ⋈ S (Email = ForEmail))

```
SELECT Landlord.Email
FROM
Landlord INNER JOIN
(
        SELECT AVG(Stars) AS TotalRating, ForEmail
        FROM UserRates NATURAL JOIN Landlord
        GROUP BY ForEmail
) AS Ratings
ON Ratings.ForEmail = Landlord.Email
ORDER BY Ratings.TotalRating DESC
```

```
LIMIT 1;
```

15. Select the tenant who rates all landlords in Bakersfield, CA.

```
SELECT Tenant.* FROM Tenant NATURAL JOIN
(
        SELECT UserRates.UEmail, ForEmail
        FROM UserRates NATURAL JOIN
(
        SELECT Property.LEmail
        FROM Property
        WHERE Property.State = "California" AND Property.City =
"Bakersfield"
) AS ForEmail
);
```

16. Select landlord's who own at least one of every property type.

```
SELECT Landlord.*
FROM Landlord NATURAL JOIN Property
WHERE Property.PropertyID IN
(
        SELECT PropertyID
        FROM Property NATURAL JOIN PropertyType
        GROUP BY PropertyID
        HAVING COUNT(DISTINCT Type) = (SELECT COUNT(DISTINCT Type)
FROM PropertyType)
);
```

Programming Logic for SQL

3.1 - Views

1. A view for a join between two tables.

```
// View landlords rated from highest to lowest
```

```
DROP VIEW IF EXISTS LHighToLow;
CREATE VIEW LHighToLow AS
SELECT User.FName, User.MI, User.LName, User.Email
FROM User INNER JOIN
(
        SELECT AVG(Stars) AS TotalRating, ForEmail
        FROM UserRates NATURAL JOIN Landlord
        GROUP BY ForEmail
) AS Ratings
ON Ratings.ForEmail = User.Email
ORDER BY Ratings.TotalRating DESC
LIMIT 10;
```

SELECT * FROM LHighToLow;

2. A view for a join between three tables.

// View recent comments

```
DROP VIEW IF EXISTS RecentComment;
CREATE VIEW RecentComment AS
SELECT MAX(Comment.Date), User.FName, Comment.Message,
CommentRates.Rating
FROM User NATURAL JOIN Comment NATURAL JOIN CommentRates;
```

OR

```
CREATE VIEW RecentComment AS
SELECT Comment.*
FROM User NATURAL JOIN Comment NATURAL JOIN CommentRates
GROUP BY Comment.Date;
```

OR

```
CREATE VIEW RecentComment AS
SELECT Comment.* FROM Comment
ORDER BY Comment.Date DESC;
```

SELECT * FROM CHighToLow;

// View top rated comments (most helpful comments)

```
DROP VIEW IF EXISTS CHighToLow;

CREATE VIEW CHighToLow AS

SELECT Comment.*

FROM Comment INNER JOIN

(

        SELECT SUM(Rating) AS TotalRating, CommentID

        FROM CommentRates NATURAL JOIN Comment

        GROUP BY CommentID

) AS Ratings

ON Ratings.CommentID = Comment.CommentID

ORDER BY Ratings.TotalRating DESC

LIMIT 5;
```

SELECT * FROM CHighToLow;

3.2 - Stored procedures/functions

1. A stored procedure for **inserting** a new record into one of your tables. The field values are passed to the procedure through the input parameters.

// User registration for a landlord account

```
DROP PROCEDURE IF EXISTS landlordRegister;

DELIMITER //

CREATE PROCEDURE landlordRegister(IN Email VARCHAR(50), IN
PhoneNumber VARCHAR(13), IN FName TEXT, IN MI TEXT, IN LName TEXT,
IN Password TEXT, IN NumOfRooms INT, IN NumOfBathrooms INT, IN Price
DECIMAL(15,2), IN State TEXT, IN City TEXT, IN Zipcode INT(5), IN Type TEXT)

BEGIN

        INSERT INTO User(Email, PhoneNumber, FName, MI, LName,
        Password) VALUES(Email, PhoneNumber, FName, MI, LName,
        Password);

        INSERT INTO Landlord(Email) VALUES(Email);
```

```
        INSERT INTO Property(LEmail, NumOfRooms, NumOfBathrooms, Price,
        State, City, Zipcode) VALUES(Email, NumOfRooms, NumOfBathrooms,
        Price, State, City, Zipcode);

        INSERT INTO PropertyType(PropertyID, Type)
        VALUES(LAST_INSERT_ID(), Type);

END //

DELIMITER ;
```

CALL landlordRegister('bob_wayne421@outlook.com', (432)413-4812, 'Bobby', 'Robert', 'Kindhorn', 'UX24zdql', 4, 2, 320.00, 'California', 'Bakersfield', 93305, 'House');

// User registration for a tenant account

```
DROP PROCEDURE IF EXISTS tenantRegister;

DELIMITER //

CREATE PROCEDURE tenantRegister(IN Email VARCHAR(50), IN
PhoneNumber VARCHAR(13), IN FName TEXT, IN MI TEXT, IN LName TEXT,
IN Password TEXT)

BEGIN

        INSERT INTO User(Email, PhoneNumber, FName, MI, LName,
        Password)VALUES(Email, PhoneNumber, FName,  MI,  LName,
        Password);

        INSERT INTO Tenant(Email)VALUES(Email);

END //

DELIMITER ;
```

CALL tenantRegister ('example@outlook.com', '(432)413-4812', 'Eada', 'M', 'Smith', 'password');

// Add a comment

```
DROP PROCEDURE IF EXISTS addComment;

DELIMITER //

CREATE PROCEDURE addComment(IN UEmail varchar(50), IN ForEmail
varchar(50), IN Message varchar(500))

BEGIN

        INSERT INTO Comment(UEmail, ForEmail, Message, Date)
```

```
        VALUES(UEmail, ForEmail, Message, NOW());
END//

DELIMITER ;
```

CALL addComment('acartmell0@loc.gov', 'ljiranek7@imageshack.us', 'Venita the reason you were evicted is because you never paid the rent on time!');

// Add occupant (Update Occupies Table)

```
DROP PROCEDURE IF EXISTS addOccupant;

DELIMITER //

CREATE PROCEDURE addOccupant(IN TEmail varchar(50), IN PropertyID BIGINT, IN Start DATE, IN End Date, IN LEmail varchar(50), OUT outCome INT)

BEGIN

        SELECT COUNT(*) INTO @propertyOwner

        FROM Property

        WHERE Property.LEmail = LEmail AND Property.PropertyID = PropertyID;

        SELECT @propertyOwner INTO outCome;

        IF @propertyOwner > 0 THEN

                INSERT INTO Occupies(TEmail, PropertyID, Start, End)

                VALUES(TEmail, PropertyID, Start, End);

        END IF;

END //

DELIMITER ;
```

CALL addOccupant('rboate5@webeden.co.uk', 100000000, ‘2005-10-02’, ‘2011-01-03’);

// Add property

```
DROP PROCEDURE IF EXISTS addProperty;

DELIMITER //

CREATE PROCEDURE addProperty(IN LEmail VARCHAR(50), IN NumOfRooms INT, IN NumOfBathrooms INT, IN Price DECIMAL(15,2), IN State TEXT, IN City TEXT, IN Zipcode INT(5), IN Type TEXT)
```

```
BEGIN

        INSERT INTO Property(LEmail, NumOfRooms, NumOfBathrooms, Price,
        State, City, Zipcode) VALUES(LEmail, NumOfRooms, NumOfBathrooms,
        Price, State, City, Zipcode);

        INSERT INTO PropertyType(PropertyID, Type)
        VALUES(LAST_INSERT_ID(), Type);

END //

DELIMITER ;
```

CALL addProperty('rboate5@webeden.co.uk', 3, 2, 260.00, 'California', 'Bakersfield', 93306, 'Apartment');

// User ratings (T and L / L and T)

```
DROP PROCEDURE IF EXISTS accountRating;

DELIMITER //

CREATE PROCEDURE accountRating(IN UEmail varchar(50), IN ForEmail
varchar(50), IN Stars TINYINT)

BEGIN

        SELECT COUNT(*) INTO @ratedBefore

        FROM UserRates

        WHERE UserRates.UEmail = UEmail AND UserRates.ForEmail =
ForEmail;

        IF @ratedBefore > 0 THEN

                UPDATE UserRates SET Stars = Stars WHERE UEmail = UEmail
        AND ForEmail = ForEmail;

        ELSE

                INSERT INTO UserRates (UEmail, ForEmail, Stars) VALUES
        (UEmail, ForEmail, Stars);

        END IF;

END //

DELIMITER ;
```

CALL accountRating ('bob_wayne421@outlook.com', 'acartmell0@loc.gov', 5);

CALL accountRating ('bob_wayne421@outlook.com', 'rboate5@webeden.co.uk', 5);

// Update property information

```
DROP PROCEDURE IF EXISTS updateProperty;

DELIMITER //

CREATE PROCEDURE updateProperty(IN LEmail VARCHAR(50), IN PropertyID
BIGINT, IN NumOfRooms INT, IN NumOfBathrooms INT, IN Price
DECIMAL(15,2), IN State TEXT, IN City TEXT, IN Zipcode INT(5), IN Type TEXT)

BEGIN

        UPDATE Property SET Property.NumOfRooms = NumOfRooms,
        Property.NumOfBathrooms = NumOfBathrooms, Property.Price =
        Price, Property.State = State, Property.City = City, Property.Zipcode
        = Zipcode WHERE Property.LEmail = LEmail  AND
        Property.PropertyID = PropertyID;

        UPDATE PropertyType SET PropertyType.Type = Type WHERE
        PropertyType.PropertyID = PropertyID;

END //

DELIMITER ;
```

CALL updateProperty('acartmell0@loc.gov', 100000001, 1, 1, 164.00, 'CA', 'Bakers', 12345, 'House');

// Update tenant information

```
DROP PROCEDURE IF EXISTS updateOccupies;

DELIMITER //

CREATE PROCEDURE updateOccupies(IN TEmail VARCHAR(50), IN
PropertyID BIGINT,IN Start DATE, IN End DATE, IN Stars TINYINT)

BEGIN

    IF Stars IS NULL THEN

        SELECT Occupies.TEmail INTO @theTenant

        FROM Occupies WHERE Occupies.TEmail = TEmail AND
        Occupies.PropertyID = PropertyID AND Occupies.Stars IS NULL;

        UPDATE Occupies SET Occupies.TEmail = TEmail, Occupies.Start
        = Start, Occupies.End = End WHERE Occupies.PropertyID
```

```
            =PropertyID AND Occupies.Stars IS NULL AND Occupies.TEmail =
            @theTenant;

        ELSE

            SELECT Occupies.TEmail INTO @theTenant

            FROM Occupies WHERE Occupies.TEmaill = TEmail AND
            Occupies.PropertyID = PropertyID AND Occupies.Stars = Stars;

            UPDATE Occupies SET Occupies.TEmail = TEmail, Occupies.Start
            = Start, Occupies.End = End WHERE Occupies.PropertyID
            =PropertyID AND Occupies.Stars = Stars AND Occupies.TEmail =
            @theTenant;

        END IF;

END //

DELIMITER ;
```

Call updateOccupies('rboate5@webeden.co.uk', 100000001, '2022-04-10', NULL, NULL);

// Give star rating for property

```
DROP PROCEDURE IF EXISTS propertyRating;

DELIMITER //

CREATE PROCEDURE propertyRating(IN TEmail VARCHAR(50), IN PropertyID
BIGINT,IN Start DATE, IN End DATE, IN Stars TINYINT)

BEGIN

        IF End IS NULL THEN

            UPDATE Occupies SET Occupies.Stars = Stars WHERE
            Occupies.PropertyID = PropertyID AND Occupies.TEmail = TEmail
            AND Occupies.Start = Start AND Occupies.End IS NULL;

        ELSE

            UPDATE Occupies SET Occupies.Stars = Stars WHERE
            Occupies.PropertyID = PropertyID AND Occupies.TEmail = TEmail
            AND Occupies.Start = Start AND Occupies.End = End;

        END IF;

END //

DELIMITER ;
```

CALL propertyRating('ljiranek7@imageshack.us', 100000001, '2022-04-30', NULL, 1);

2. A stored procedure for **deleting an existing record based on the primary key** of your selected table.

```
// Deleting an existing tenant account

DROP PROCEDURE IF EXISTS deleteTenantAcc;

DELIMITER //

CREATE PROCEDURE deleteTenantAcc(IN TEmail varchar(50))

BEGIN

        SELECT Email INTO @tenantEmail

        FROM Tenant

        WHERE Email = TEmail;

        DELETE FROM UserRates WHERE UEmail = @tenantEmail;

        DELETE FROM UserRates WHERE ForEmail = @tenantEmail;

        DELETE FROM CommentRates WHERE UEmail = @tenantEmail;

        DELETE FROM Comment WHERE UEmail = @tenantEmail;

        DELETE FROM Comment WHERE ForEmail = @tenantEmail;

        DELETE FROM Tenant WHERE Email = @tenatEmail;

        DELETE FROM User WHERE Email = @tenantEmail;

END//

DELIMITER ;
```

CALL deleteTenantAcc('rboate5@webeden.co.uk');

```
// Deleting an existing landlord account

DROP PROCEDURE IF EXISTS deleteLandlord;

DELIMITER //

CREATE PROCEDURE deleteLandlord(IN LEmail varchar(50))

BEGIN

        SELECT Email INTO @landlordEmail
```

```
        FROM Landlord
        WHERE Email = LEmail;
        DELETE FROM UserRates WHERE UEmail = @landlordEmail;
        DELETE FROM UserRates WHERE ForEmail = @landlordEmail;
        DELETE FROM CommentRates WHERE UEmail = @landlordEmail;
        DELETE FROM Comment WHERE UEmail = @landlordEmail;
        DELETE FROM Comment WHERE ForEmail = @landlordEmail;
        DELETE FROM Landlord WHERE Email = @landlordEmail;
        DELETE FROM User WHERE Email = @landlordEmail;
END//
DELIMITER ;
CALL deleteLandlord('ehallums8@go.com');
// Delete tenant who is a renter from Occupies
DROP PROCEDURE IF EXISTS deleteTenant;
DELIMITER //
CREATE PROCEDURE deleteTenant(IN TEmail VARCHAR(50), IN PropertyID
BIGINT,IN Start DATE, IN End DATE, IN Stars TINYINT)
BEGIN
        IF Stars IS NULL THEN
                IF End IS NULL THEN
                        DELETE FROM Occupies WHERE Occupies.TEmail =
                        TEmail AND Occupies.Start = Start AND
                        Occupies.PropertyID = PropertyID AND Occupies.End IS
                        NULL AND Occupies.Stars IS NULL;
                ELSE
                        DELETE FROM Occupies WHERE Occupies.TEmail =
                        TEmail AND Occupies.Start = Start AND
                        Occupies.PropertyID = PropertyID AND Occupies.End = End
                        AND Occupies.Stars IS NULL;
                END IF;
```

```
        ELSE
            IF End IS NULL THEN
                DELETE FROM Occupies WHERE Occupies.TEmail =
                TEmail AND Occupies.Start = Start AND
                Occupies.PropertyID = PropertyID AND Occupies.End IS
                NULL AND Occupies.Stars = Stars;
            ELSE
                DELETE FROM Occupies WHERE Occupies.TEmail =
                TEmail AND Occupies.Start = Start AND
                Occupies.PropertyID = PropertyID AND Occupies.End = End
                AND Occupies.Stars = Stars;
            END IF;
        END IF;
END //

DELIMITER ;
```

Call deleteTenant('ljiranek7@imageshack.us', 100000002, '2022-04-30', NULL, NULL);

// Deleting an existing property

```
DROP PROCEDURE IF EXISTS deleteProperty;

DELIMITER //

CREATE PROCEDURE deleteProperty(IN LEmail VARCHAR(50), IN PropertyID
BIGINT, OUT outCome INT)

BEGIN
        SELECT COUNT(*) INTO @numOfProperties
        FROM Property
        WHERE Property.LEmail = LEmail;
        SELECT @numOfProperties INTO outCome;
        IF @numOfProperties > 1 THEN
            DELETE FROM Property WHERE Property.LEmail = LEmail AND
            Property.PropertyID = PropertyID;
        END IF;
```

END//

DELIMITER ;

CALL deleteProperty(LEmail, PropertyID, $outCome);

3. A stored procedure which **returns statistical metrics for a table over a period of time**: average sales of the last month, inventory item most ordered, highest-paying customer, etc.

// View all owners that own properties with a certain number of bedrooms

DROP PROCEDURE IF EXISTS bedRooms;

DELIMITER //

CREATE PROCEDURE bedRooms(IN numRooms INT)

BEGIN

SELECT DISTINCT User.FName, User.MI, User.LName, User.Email, NumOfRooms

FROM Landlord NATURAL JOIN User NATURAL JOIN Property

WHERE NumOfRooms >= numRooms;

END//

DELIMITER ;

CALL bedRooms(2);

CALL bedRooms(4);

CALL bedRooms(3);

3.3 - Triggers

In this section, you will implement 3 triggers:

1. One for deleting a row from a table

// Deleting an existing account (Landlord/Tenant)

```
DROP TABLE IF EXISTS oldAccount;
CREATE TABLE oldAccount
(
        Email VARCHAR(50) NOT NULL,
        PhoneNumber VARCHAR(13) NOT NULL,
        FName TEXT NOT NULL,
        MI TEXT,
```

```
        LName TEXT NOT NULL,
        Password TEXT NOT NULL,
        UserType TEXT NOT NULL,
        DeletedOn DATETIME DEFAULT CURRENT_TIMESTAMP,
        PRIMARY KEY(Email)
);
DROP TRIGGER IF EXISTS deleteAccount;
DELIMITER //
CREATE TRIGGER deleteAccount
BEFORE DELETE ON User
FOR EACH ROW
BEGIN
        SET @accType = 'Tenant';

        SELECT COUNT(*) INTO @hasDeleted
        FROM oldAccount
        WHERE Email = OLD.Email;

        SELECT COUNT(*) INTO @accCheck
        FROM Landlord
        WHERE Email = OLD.Email;

        IF @accCheck > 0 THEN
                SET @accType = 'Landlord';
        END IF;
        IF @hasDeleted > 0 THEN
                UPDATE oldAccount SET Email = OLD.Email, PhoneNumber =
        OLD.PhoneNumber , FName = OLD.FName, MI = OLD.MI, LName =
        OLD.LName, Password = OLD.Password, UserType =@accType,
        DeletedOn = NOW() WHERE Email = OLD.Email;
        ELSE
                INSERT INTO oldAccount (Email, PhoneNumber, FName, MI,
        LName, Password, UserType)
                VALUES (OLD.Email, OLD.PhoneNumber, OLD.FName, OLD.MI,
        OLD.LName, OLD.Password, @accType);
        END IF;
END //
DELIMITER ;
CALL deleteLandlord('acartmell0@loc.gov');
SELECT * FROM Landlord;
```

SELECT * FROM User;
SELECT * FROM oldAccount;

CALL deleteTenant('ljiranek7@imageshack.us');
SELECT * FROM Tenant;
SELECT * FROM User;
SELECT * FROM oldAccount;

2. One for updating a row in a table

// Updating comment message

```
DROP TABLE IF EXISTS editComment;

CREATE TABLE editComment

(

        CommentID SERIAL,

        UEmail VARCHAR(50) NOT NULL,

        ForEmail VARCHAR(50) NOT NULL,

        oldMessage VARCHAR(500) NOT NULL,

        newMessage VARCHAR(500) NOT NULL,

        updateTime DATETIME DEFAULT CURRENT_TIMESTAMP,

        PRIMARY KEY(CommentID),

        FOREIGN KEY(CommentID) REFERENCES Comment(CommentID) ON
DELETE CASCADE ON UPDATE CASCADE

);

DROP TRIGGER IF EXISTS editMsg;

DELIMITER //

CREATE TRIGGER editMsg

BEFORE UPDATE ON Comment

FOR EACH ROW

BEGIN

        INSERT INTO editComment (CommentID, UEmail, ForEmail,
oldMessage, newMessage, Date)
```

```
        VALUES(OLD.CommentID, OLD.UEmail, OLD.ForEmail, OLD.Message,
NEW.Message, NOW());

END //

DELIMITER ;

UPDATE Comment SET Message = 'Nevermind, they were bad.' WHERE
Order_ID = '200000000';

SELECT * FROM Comment;

// Updating account information

DROP TABLE IF EXISTS oldUsers;

CREATE TABLE oldUsers

(

        Email VARCHAR(50) NOT NULL,

        PhoneNumber VARCHAR(13) NOT NULL UNIQUE,

        FName TEXT NOT NULL,

        MI TEXT,

        LName TEXT NOT NULL,

        Password TEXT NOT NULL,

        updatedTime DATETIME DEFAULT CURRENT_TIMESTAMP,

        PRIMARY KEY(Email),

        FOREIGN KEY(Email) REFERENCES User(Email) ON DELETE
CASCADE ON UPDATE CASCADE

);

DROP TRIGGER IF EXISTS updateUser;

DELIMITER //

CREATE TRIGGER updateUser

BEFORE UPDATE ON User

FOR EACH ROW

BEGIN
```

```
        SELECT COUNT(*) INTO @hasUpdated

        FROM oldUsers

        WHERE Email = OLD.Email;

        IF @hasUpdated > 0 THEN

            UPDATE oldUsers SET PhoneNumber =OLD.PhoneNumber,
        FName =OLD.FName, MI =OLD.MI, LName =OLD.LName, Password
        =OLD.Password, updatedTime =NOW() WHERE Email =OLD.Email;

        ELSE

            INSERT INTO oldUsers

            VALUES(OLD.Email, OLD.PhoneNumber, OLD.FName, OLD.MI,
        OLD.LName, OLD.Password, NOW());

        END IF;
END //

DELIMITER ;
```

UPDATE User SET PhoneNumber = '(100)555-7334', FName = 'Emyle', WHERE Email = 'ehallums8@go.com';

UPDATE User SET PhoneNumber = '(943)766-2582', FName = 'Madelaine' WHERE Email = 'rboate5@webeden.co.uk';

SELECT * FROM User;

3. One for inserting a row into a table.

// Creating a new account

```
DROP TABLE IF EXISTS newAccMsg;

CREATE TABLE newAccMsg

(

        Email VARCHAR(50),

        FName TEXT,

        Message TEXT,

        PRIMARY KEY(Email),

        FOREIGN KEY(Email) REFERENCES User(Email) ON DELETE
CASCADE ON UPDATE CASCADE
```

```
);

DROP TRIGGER IF EXISTS newAcc;

DELIMITER //

CREATE TRIGGER newAcc

AFTER INSERT ON User

FOR EACH ROW

BEGIN

        INSERT INTO newAccMsg(Email, FName, Message)

        VALUES(NEW.Email, NEW.FName, CONCAT('Your account was created
successfully. Welcome ', NEW.FName, '!') );

END //

DELIMITER ;
```

CALL landlordRegister('leny@example.com', (432)516-9875, 'leny', ' ', 'leny', 'test', 100000004, 3, 2, 224.00, 'California', 'Bakersfield', 93306, 'House');

SELECT * FROM newAccMsg;

// Landlord rates Tenant / Tenant rates Landlord (New ratings for account)

```
DROP TRIGGER IF EXISTS ratingAcc;
DELIMITER //
CREATE TRIGGER ratingAcc
BEFORE INSERT ON UserRates
FOR EACH ROW
BEGIN
        SELECT COUNT(*) INTO @isUserLandlord
        FROM Landlord
        WHERE Email = NEW.UEmail;

        SELECT COUNT(*) INTO @isUserTenant
        FROM Tenant
        WHERE Email = NEW.UEmail;

        SELECT COUNT(*) INTO @isForLandlord
        FROM Landlord
```

```
        WHERE Email = NEW.ForEmail;

        SELECT COUNT(*) INTO @isForTenant
        FROM Tenant
        WHERE Email = NEW.ForEmail;

        IF NEW.UEmail = NEW.ForEmail THEN
                SIGNAL SQLSTATE '45000' SET
                MYSQL_ERRNO = 31001,
                MESSAGE_TEXT = 'Invalid rating - no self-ratings allowed';
        END IF;

        IF  (@isUserLandlord > 0 AND @isForTenant = 0) OR
        (@isUserTenant > 0 AND @isForLandlord = 0) THEN
                SIGNAL SQLSTATE '45000' SET
                MYSQL_ERRNO = 31001,
                MESSAGE_TEXT = 'Invalid rating - must be between tenant and
        landlord or landlord and tenant';
        END IF;
END //
DELIMITER ;
```