Felicia Kuan

Lyman Shen

COEN 178

7 June 2018

Happy Renter's Rental Management System Project Documentation

This is a web application we collaborated on to complete for our Database Systems (COEN 178) final project.

The objective was to both design and implement a web application called "Happy Renter's Rental Management INC" that manages rental properties on behalf of the clients. It is a DBMS meant to assist a rental manager in keeping track of rental properties and lease agreements. Please use this documentation as a clearer means of understanding the various features of our application.

Basic Properties of the Rental Management System:

- Each branch of the Happy Renter's INC is managed by one manager. This property is enforced by primary keys for Branch ID (branchid in SQL implementation) and Manager ID (empid) within each of their respective tables, and adding a Foreign key for Branch ID within the Manager ID . The uniqueness of the IDs for branch and manager ensures that there is only one manager per branch.
- Each rental property managed by Happy Renter's is managed by one supervisor, yet a supervisor may supervise many properties. This is enforced by placing the primary key for the supervisor (empid)

- Within the database, the table that keeps track of the information for a specific rental property also records the phone number of the original property owner, information on the current renter, supervisor, address of the rental property, number of rooms, monthly rent, and status that can be "available" or "leased." This also contains the rental property's ID number, which is its primary key, as well as the rental property's start date of availability.
- Basic information on property owners and renters are also stored in the database, in separate tables, with their phone numbers as the primary keys.

Entity Relationship (ER) Model

For the ER Model, please refer to Figures 1 and 2 outside of this file for better visualization of the model.

Brief description of the model: There are six entities in this model (seven if you use an object-oriented approach to handling subclasses). Only one of them is a weak entity and the others have their own primary keys and can exist on their own. The relationships are stated below, and we also included specifics about how we linked entities with relationships together, such as how we made the primary key from one entity a foreign attribute of another. All these details are stated below.

Branch and **Manage**r have a one to one relationship

Therefore, the primary key for branch (branchid) is a foreign attribute of Manager

Supervisor and **Rental_Property** have a one to many relationship

So, primary key of Supervisor(empid) is attribute in Rental_Property

Rental_Property and **Property_Owner** have a many to one relationship

So, primary key of **Property_owner** (phone) is an attribute in **Rental_Property**

Renter and **Rental_Property** are in a many to many relationship and **Lease_Agreement** is their relationship.

Primary keys of **Renter** (work_phone) and **Rental_Property** (rental_num) are defined inside **Lease_Agreement**

For additional information on the ER Models and how the attributes are named and created, you can consult the file named “Schema For Happy Renter’s.”

Functional Dependencies (FDs)

The functional dependencies we used for this project are as follows:

- Branchid → Manager.empid
- Manager.empid → branchid
    - The FDs above are enforced by the fact that they’re primary keys
- Supervisor.empid → Manager.empid (one manager for many supervisors)
- Rental_PropertyID (rental_num) → Supervisor.empid
- Rental_Property.rental_num → Property_Owner.work_phone
- Rental_Property.rental_num, Rental_Property.startDate → Lease_Agreement

To refer to how we organized all of our data into tables or if you need more detail on the naming schemes for these attributes can be referred to in the Schema(pdf) or Create_tables.sql file.

Queries, Possible Test Values, and Results

The following are listings of the queries representing the different transactions our system is supposed to come equipped with. For the sake of simplicity, if the queries require user input,

we provided a sample user input highlighted in yellow. Also, these queries operate under the assumption that there are entries within the database to work and that the user inputs information that fall within the constraints (see next section). The resulting table from the query is found in our spool file titled “results_from_queries.txt.”

1. Generate list of rental properties available for a specific branch along with the manager's name.

```
SELECT rental_num, Street, City, Zip, name
FROM Rental_Property, Employee
WHERE Rental_Property.empId IN (Select empId From Supervisor
Where managerId = (Select empid from Manager
where branchId =  'b0112'
AND Rental_Property.empId = Employee.empId
AND status = 'available';
```

2. Generate list of supervisors and the properties (with address) they supervise */

```
SELECT Supervisor.empId, rental_num, Street, City, Zip
FROM Rental_Property, Supervisor
WHERE Supervisor.empId = Rental_Property.empId;
```

3. Generate a list of rental properties by a specific owner, listed in a HappyRenter’s branch

```
SELECT rental_num, Street, City, Zip
FROM Rental_Property
WHERE owner_phone = '7682229997';
```

4. Show a listing of properties available, where the properties should satisfy the criteria (city, no of rooms and/or range for rent given as input).

```
SELECT rental_num, start_date_of_availibility
FROM Rental_Property
WHERE city = 'BKNY' AND num_rooms = 2
AND monthly_rent < 1000 AND monthly_rent > 0;
```

5. Show the number of properties available for rent.

```
SELECT count(*) AS Available_Property
FROM Rental_Property
WHERE status = 'available';
```

6. Create Lease Agreement using GUI

Note: The lease agreement form operates under the assumption that appropriate profiles for renters and property owners were created, and rental properties were registered, prior to attempting to create a new lease agreement. Thus, the user, while filling out the lease agreement, has to be aware which properties are available during which dates. Without completing the previous tasks, the lease agreement would not successfully be created.

Our GUI is found here: students.engr.scu.edu/~lshen/RentalManagementSystem

7. Show a lease agreement for a renter.

```
SELECT name, work_phone, rental_num, start_date, end_date,
        deposit_amt, rent_amt, sup_name
FROM Lease_Agreement, Renter
WHERE renter_wphone = work_phone AND
```

renter_wphone = '2405409801';

8. Show the renters who rented more than one rental property.

```
SELECT * FROM Renter
WHERE work_phone IN (Select distinct renter_wphone
from Lease_Agreement
group by renter_wphone having count(*) > 1);
```

9. Show the average rent for available properties in a town (name of the town is entered as input)

```
SELECT Avg(monthly_rent) AS Average_Rent
FROM Rental_Property
WHERE City = 'BKNY' AND status = 'available';
```

10. Show the names and addresses of properties whose leases will expire in next two months (from the current date).

```
SELECT Rental_Property.rental_num, Street, City, Zip, end_date
FROM Rental_Property, Lease_Agreement
where Rental_Property.rental_num = Lease_Agreement.rental_num
AND end_date <= trunc(SYSDATE + 60)
AND end_date >= trunc(SYSDATE);
```

As a side note, although these queries were executed in SQL, we also enabled them to be executed on the GUI as well, so you may test them on the GUI if you'd like.

Link to GUI: students.engr.scu.edu/~lshen/RentalManagementSystem

Constraints

As a precaution to ensure the accuracy and integrity of our data, we enforced the following constraints to prevent users from randomly input data:

- Lease agreements must be made for a minimum of 6 months and a maximum of a year.
    - We achieved this by creating a CHECK constraint that the difference in months between start_date and end_date in Lease_Agreements is between 6 and 12, inclusive.
- We made sure to update our records and change the status of a property to "leased" once a lease agreement for that property has been made.
    - Used a trigger to achieve this.
- A rental property is removed from a supervisor's list of rental properties once the owner decides not to rent the property out anymore
    - System automatically implements this due to the way we organized the attributes in each table
- The rent of a property is increased by 10% with every new lease on the same property.
    - Created a compound trigger to achieve this