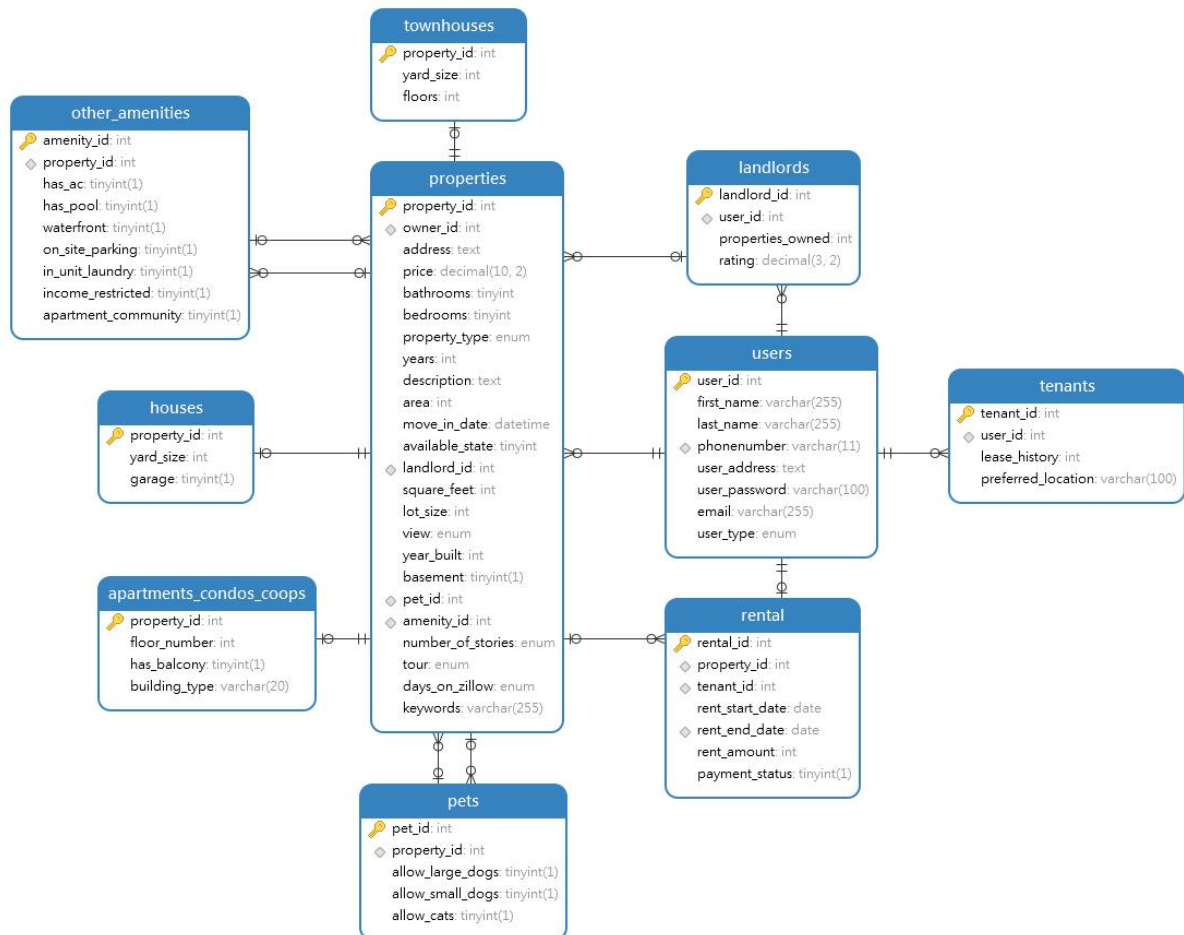


Team 6 House Rental Database Management Systems

1.Final E-R Diagram



2. Database overview

The screenshot shows a database management interface. On the left, a sidebar lists the database structure: 'damg' contains 'Tables' (apartments_condos_coops, houses, landlords, other_amenities, pets, properties, rental, tenants, townhouses, users) and 'Views' (availableproperties, landlordinfo, tenantinfo). Below these are 'Functions' (CalculateAveragePriceByType, GetAllLandlords, GetAllTenants, GetFacilityInfo, UpdateUserPasswordWithHash) and 'Queries' and 'Backups'. The main area displays the 'users' table with columns: user_id, first_name, last_name, phonenumber, user_address, user_password, email, and user_type. The table contains 20 rows of data, with the first row highlighted.

user_id	first_name	last_name	phonenumber	user_address	user_password	email	user_type
1	Alice	Smith	1234567890	123 Street A	c822a0abf4ef0a5fc2a4c2c	alice@example.com	tenant
2	Bob	Johnson	1234567891	124 Street B	f3ce7c47a12cd02443eac	bob@example.com	landlord
3	Emma	Johnson	1234567892	125 Street C	password3	emma@example.com	tenant
4	James	Williams	1234567893	126 Street D	password4	james@example.com	landlord
5	Olivia	Brown	1234567894	127 Street E	password5	olivia@example.com	tenant
6	William	Jones	1234567895	128 Street F	password6	william@example.com	landlord
7	Ava	Garcia	1234567896	129 Street G	password7	ava@example.com	tenant
8	Benjamin	Miller	1234567897	130 Street H	password8	benjamin@example.com	landlord
9	Sophia	Davis	1234567898	131 Street I	password9	sophia@example.com	tenant
10	Lucas	Martinez	1234567899	132 Street J	password10	lucas@example.com	landlord
11	Mia	Hernandez	1234567900	133 Street K	password11	mia@example.com	tenant
12	Henry	Lopez	1234567801	134 Street L	password12	henry@example.com	landlord
13	Ella	Gonzalez	1234567802	135 Street M	password13	ella@example.com	tenant
14	Alexander	Wilson	1234567803	136 Street N	password14	alexander@example.com	landlord
15	Isabella	Anderson	1234567804	137 Street O	password15	isabella@example.com	tenant
16	Daniel	Thomas	1234567805	138 Street P	password16	daniel@example.com	landlord
17	Grace	Taylor	1234567806	139 Street Q	password17	grace@example.com	tenant
18	Jackson	Moore	1234567807	140 Street R	password18	jackson@example.com	landlord
19	Emily	Jackson	1234567808	141 Street S	password19	emily@example.com	tenant
20	Michael	Martin	1234567809	142 Street T	password20	michael@example.com	landlord

3. The Views

The screenshot shows a database management interface with a sidebar listing the database structure. The 'Views' section is expanded, showing a list of views: availableproperties, landlordinfo, and tenantinfo. The 'availableproperties' view is selected and highlighted.

Views
availableproperties
landlordinfo
tenantinfo

property_id	owner_id	address	price	bathrooms	bedrooms	property_type	years	description	area	m
3	3	789 Hill Rd	350000.00	3	4	townhouses	8	Spacious townhouse with	2500	20
5	5	202 Forest Rd	280000.00	2	2	houses	12	Charming house near the	1800	20
7	7	404 Oak St	310000.00	3	3	townhouses	9	Townhouse in a family-frie	2100	20
9	9	606 Beach Rd	470000.00	3	4	houses	14	Beachfront house with stu	2700	20

first_name	last_name	landlord_id	user_id	properties_owned	rating
Bob	Johnson	2	2	3	3.00
William	Jones	4	6	6	4.20
Benjamin	Miller	5	8	2	4.00
Lucas	Martinez	6	10	8	4.60
Henry	Lopez	7	12	3	3.50
Alexander	Wilson	8	14	5	4.10
Daniel	Thomas	9	16	7	4.30
Jackson	Moore	10	18	4	3.90

first_name	last_name	tenant_id	user_id	lease_history	preferred_location
Emma	Johnson	1	3	1	Downtown
Ava	Garcia	2	7	2	Suburbs
Sophia	Davis	3	9	3	Near University
Mia	Hernandez	4	11	4	City Center
Ella	Gonzalez	5	13	1	Riverside
Isabella	Anderson	6	15	2	Uptown
Grace	Taylor	7	17	3	East Side
Emily	Jackson	8	19	1	West End

4. Data encryption

```
-- -----  
-- Procedure structure for UpdateUserPasswordWithHash  
-- -----  
DROP PROCEDURE IF EXISTS `UpdateUserPasswordWithHash`;  
delimiter ;;  
CREATE PROCEDURE `UpdateUserPasswordWithHash`(IN p_user_id INT, IN p_new_password VARCHAR(100))  
BEGIN DECLARE hashed_password VARCHAR(64);  
    SET hashed_password = SHA2(p_new_password, 256);  
    UPDATE Users SET user_password = hashed_password WHERE user_id = p_user_id;  
    END  
;;  
delimiter ;
```

We use SHA-256 algorithm to encrypt users' password. And we create a trigger, after insert user' information, trigger will automatically use this store procedure to encrypt.

5. Store procedure

```
-- -----  
-- Procedure structure for GetAllLandlords  
-- -----  
DROP PROCEDURE IF EXISTS `GetAllLandlords`;  
delimiter ;;  
CREATE PROCEDURE `GetAllLandlords`()  
BEGIN  
    SELECT Users.username, Landlords.* FROM Users JOIN Landlords ON Users.user_id = Landlords.user_id  
    WHERE Users.user_type = 'landlord';  
  
    END  
;;  
delimiter ;
```

```
-- -----  
-- Procedure structure for UpdateUserPasswordWithHash  
-- -----  
DROP PROCEDURE IF EXISTS `UpdateUserPasswordWithHash`;  
delimiter ;;  
CREATE PROCEDURE `UpdateUserPasswordWithHash`(IN p_user_id INT, IN p_new_password VARCHAR(100))  
BEGIN DECLARE hashed_password VARCHAR(64);  
    SET hashed_password = SHA2(p_new_password, 256);  
    UPDATE Users SET user_password = hashed_password WHERE user_id = p_user_id;  
    END  
;;  
delimiter ;
```

```
-- -----  
-- Procedure structure for GetAllTenants  
-- -----  
DROP PROCEDURE IF EXISTS `GetAllTenants`;  
delimiter ;;  
CREATE PROCEDURE `GetAllTenants`()  
BEGIN  
    SELECT Users.username, Tenants.* FROM Users JOIN Tenants ON Users.user_id = Tenants.user_id  
    WHERE Users.user_type = 'tenant';  
    END  
;;  
delimiter ;
```

We create 3 store procedures to help us manage this database better.

6. Triggers

```
-- -----  
-- Triggers structure for table Rental  
-- -----  
DROP TRIGGER IF EXISTS `UpdatePropertyAvailability`;  
delimiter ;;  
CREATE TRIGGER `UpdatePropertyAvailability` AFTER INSERT ON `Rental` FOR EACH ROW BEGIN  
UPDATE Properties SET available_state = 0 WHERE property_id = NEW.property_id;  
END  
;;  
delimiter ;
```

We also create another trigger to manage our database.

7. Table-Level Check Constraint

```
CONSTRAINT `CheckPropertyPrice` CHECK ((`price` >= 0))  
  
CONSTRAINT `CheckLeaseDates` CHECK ((`rent_start_date` < `rent_end_date`)),  
CONSTRAINT `CheckRentAmount` CHECK ((`rent_amount` > 0))
```

We create some check constraints to help us manage this database better.

8. Non-Clustered Index

Name	Fields	Index Type	Index method	Comment
userid_UNIQUE	`rental_id` ASC	UNIQUE	BTREE	
userphone_UNIQUE	`tenant_id` ASC	UNIQUE	BTREE	
houseid_UNIQUE	`rent_end_date` ASC	UNIQUE	BTREE	
property_id	`property_id` ASC	NORMAL	BTREE	

We create many non-clustered index to improve the speed of database retrieval