

# **EECS3421 Final Report**

**Title: Homes for Sale, Case 6**

# **1. Introduction and preliminary study**

## **Background of the Organization Involved**

Hamilton Area Home Listings is a real small sub-organization founded in 2019 under RE/MAX Escarpment Realty Inc. Brokerage, that specializes in real estate affairs for the Hamilton area. They primarily deal with residential real estate to provide a centralized system for home buyers and sellers. The Hamilton team consists of four people, led by Mark Silenzi of the Silenzi team. The Hamilton Area Home Listings organization's main office is situated in Burlington as part of the Silenzi team's headquarters in RE/MAX Escarpment Realty Inc. Brokerage. The website is active with roughly 2000 active listings and a blog containing information on featured listings and general real estate advice provided by the team.

## **Objectives and Scope of the Application**

The objective of the website is to provide a platform for selling and purchasing of properties in the Hamilton area of Ontario. The website's primary users are homebuyers and homesellers. The homesellers can use the website to have their property evaluated and listed for sale with the help of real estate agents. The homebuyers can use the website to search for properties in Hamilton and get in contact with the owner of the property to discuss details regarding the sale of the property. The scope of the application includes account creation, posting property listings, searching for listings based on various filters such as location and price, and messaging functionality between a potential buyer and the owner of a listing. All of these functions need to be supported by the database that will store the listings, accounts, messages between accounts and more.

## **Functional Requirements of the Application**

### User account management

The application must let users create an account with a full name, email, phone number, and password. Users will only be able to access listing details, post listings, and send messages after they have created an account and logged in. As such, the database must store all the user account information.

### Creating a listing

The application should allow logged-in users to create a new property listing. When creating a listing, the user must input all the required information for a listing, including the listing's full address, postal code, number of bedrooms and bathrooms, price, area in square feet, year built, property type, images of the property, and a description of the property. All of this information must be stored in the database for each listing. Each listing should also be associated with the user who created it and the database must reflect that.

#### Listing browsing and searching

The application must allow both guests and logged-in users to search listings. The user can specify various search filters. These filters include city, postal code, number of bedrooms and bathrooms, minimum and maximum price, area in square feet, year built, and property type.

#### Listing details page

The details page for a listing must only be viewable by a logged-in user. It must display all of the searchable attributes mentioned above in the “*Listing browsing and searching*” section, and additional information such as the property’s images and description, and the listing creator’s contact information.

#### Account’s “Favourite” page

A logged-in user must be able to save listings to their “Favourite” page. The user must be able to rate saved listings by condition, location, whether it is nearby important places, whether it is in proximity of schools, and whether the tax level is good. The user must additionally be able to save their notes about the listing.

#### Messaging system

The website should allow logged-in users to send messages to the owner of a listing. These messages should be stored in the database to ensure security between the users and for record keeping.

## **2. Database Modelling**

### **External Views**

#### User Account Creation Form (Fig. 1)

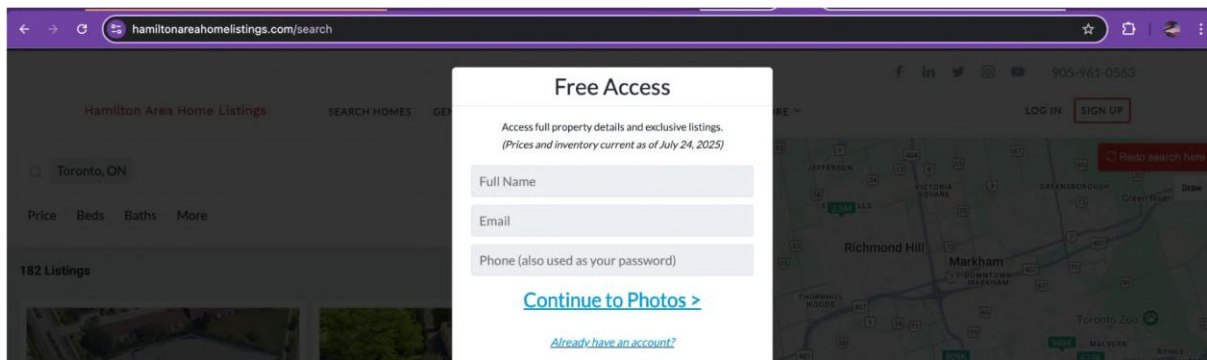


Figure 1: User Account Creation Form

Users must log in or sign up before being granted access to listing details, posting listings, and sending messages (Fig. 1). The required data for an account include:

- Full name
- Email
- Phone number (which is used as their password)

### Listing Filter and Search

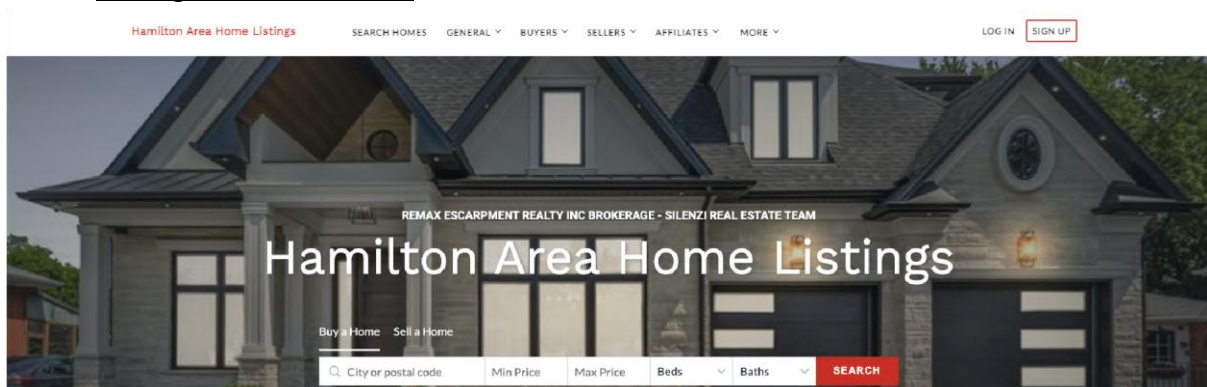
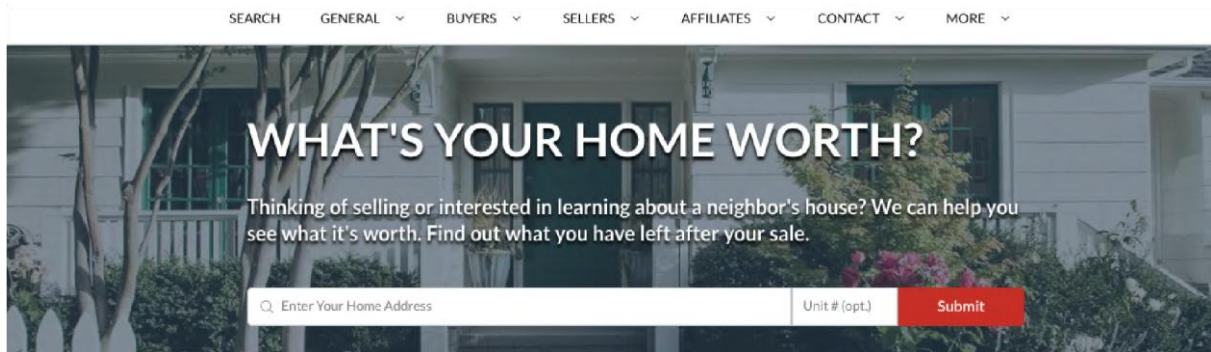


Figure 2: Listing filter and search on the home page

On the home page (Fig. 2), the user can filter and search listings by:

- A price range
- An area range in square feet
- A year range in which the property was built
- City or postal code
- Number of bedrooms and bathrooms

## Creating Listing



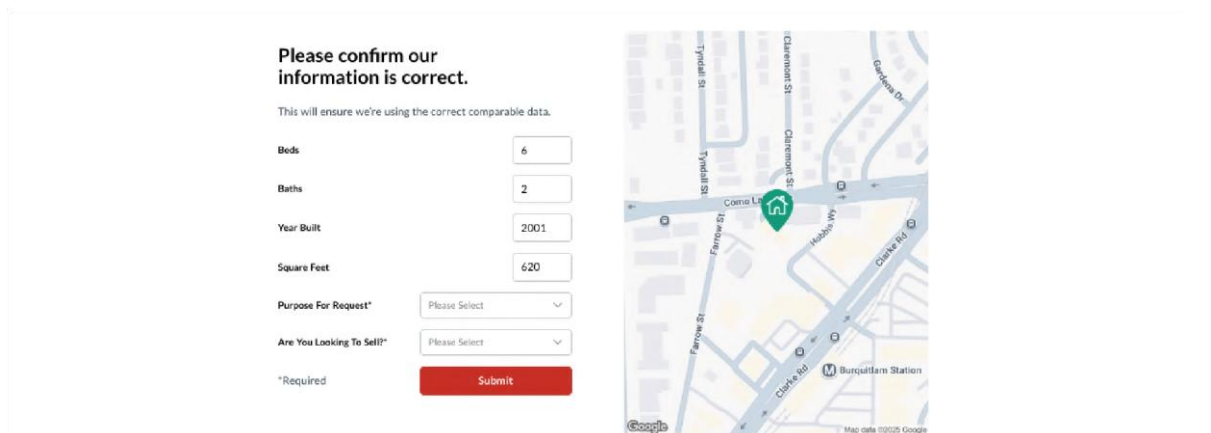
SEARCH GENERAL BUYERS SELLERS AFFILIATES CONTACT MORE

# WHAT'S YOUR HOME WORTH?

Thinking of selling or interested in learning about a neighbor's house? We can help you see what it's worth. Find out what you have left after your sale.

Q Enter Your Home Address Unit # (opt.) Submit

Figure 3: Inserting the new listing's address

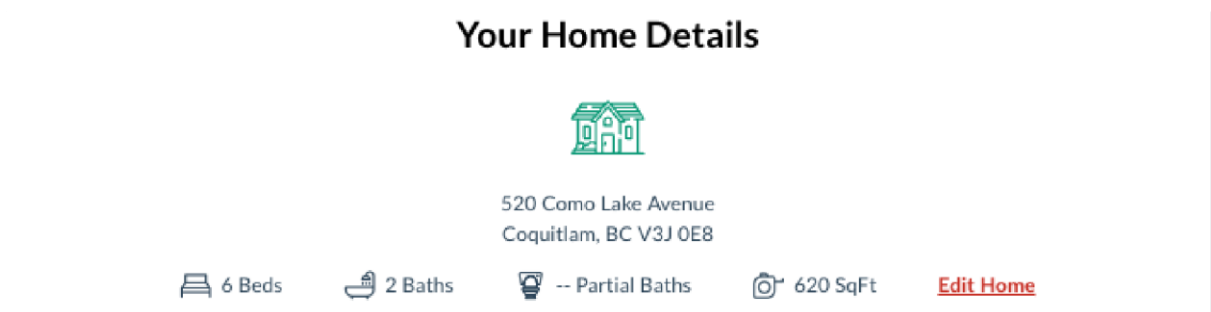


Please confirm our information is correct.  
This will ensure we're using the correct comparable data.


Beds 6  
Baths 2  
Year Built 2001  
Square Feet 620  
Purpose For Request\* Please Select  
Are You Looking To Sell?\* Please Select  
\*Required Submit

Map showing the location of the property (520 Como Lake Avenue, Coquitlam, BC V3J 0E8).

Figure 4: Inserting the new listing's details



## Your Home Details



520 Como Lake Avenue  
Coquitlam, BC V3J 0E8





 6 Beds  2 Baths  -- Partial Baths  620 SqFt [Edit Home](#)

Figure 5: The pending listing's finalized detail

The user can create a listing after providing their property's address (Fig. 3), and other required details e.g number of bedrooms, year built (Fig. 4).

## Listing Details

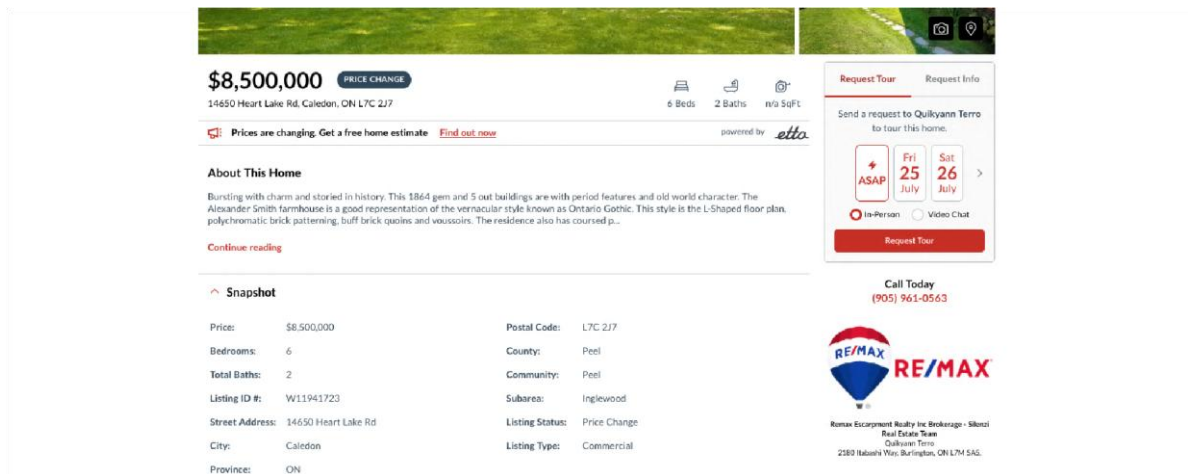


Figure 6: A sample listing's detail page

Logged-in users can view a listing's details on a separate page (Fig. 6). Its searchable attributes include:

- Price
- Address (comprised of street address, city, province, postal code)
- Number of beds
- Number of bathrooms
- Area in square feet

The listing contains other details such as the property's ID, images and description, and the listing creator's contact information.

### "Favourite" Page



Figure 7: A user adding a listing to their "Favourite" section on the listing's detail page

120 Rosedale Valley Rd Unit 201, Toronto C09, ON M4W 1P8

FULL LISTING REMOVE

1/30

**\$1,289,000**  
3 Beds, 2 Baths  
1600-1799 sqft

**SCORECARD**

	2 PROS	3 CONS
Condition	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Location	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Nearby	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Schools	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Taxes	<input type="checkbox"/>	<input checked="" type="checkbox"/>

**NOTES** 0  
View More

SAVE NOTE

Figure 8: A user's sample rating of a listing in their "Favourite" section

A logged-in user can save listings to their "Favourite" section (Fig. 7). The user can rate a saved listing in a 'scorecard' (Fig. 8) according to the following criteria:

- Condition
- Location
- Whether it is nearby important places
- Whether it is in proximity to schools
- Whether the tax level is good

The user can additionally save their notes about the listing (Fig. 8).

## Messaging System

**\$8,500,000** 6 Beds | 2 Baths | n/a sqft 14650 Heart Lake Rd

Favorite Share

**Quikyann Terro**  
Can Answer Your Questions

How much out-of-pocket expenses do I need to purchase a home? Does this home have any offers on it?

How much down-payment would I need to buy? Do repairs come out of my pocket or the seller?

Is now a good time to buy a home?

Full Name  
Hehe Haha

Email Address  
hehehaha1234@yahoo.com

Phone Number  
5471139922

Message  
I'm interested in 14650 Heart Lake Rd, Caledon, ON L7C 2J7.

Send

**Request Tour** Request Info

Send a request to Quikyann Terro to tour this home.

ASAP Fri 25 July Sat 26 July

☒ In-Person ☐ Video Chat

Request Tour

Call Today  
(905) 961-0563

**RE/MAX**

Remax Escarpment Realty Inc. Brokerage - Silent  
Real Estate Team  
Quikyann Terro  
2180 Itabashi Way, Burlington, ON L7M 5A5.

Figure 9: Messaging system on a listing's detail page

On a listing's detail page (see Fig. 6 for a sample detail page), logged-in users can send messages to the owner for inquiries (see Fig. 9).

## **Business Rules and Assumptions**

### Property Listing

- A listing is uniquely identified by one ID
- A listing is owned by one account
- A listing is can be added to the “Favourite” section of 0 or more accounts
- A listing has one address
- A listing is involved in 0 or 1 purchase
- 0 or more messages can be sent with regards to a listing

### Account

- An account has a unique ID
- An account can own 0 or more listings
- An account can add 0 or more listings in its “Favourite” section
- An account can purchase 0 or more listings
- An account can send 0 or more messages with regards to some listing - An account can store 0 or more credit cards

### Favourite

- An entry in some account’s “Favorite” section is uniquely identified by the account’s ID and the ID of the listing added
- An entry in some account’s “Favorite” section belongs to one account - An entry in some account’s “Favorite” section adds one listing

### Message

- A message is uniquely identified by its ID
- A message is sent by one account
- A message is sent with regards to one property listing



### Purchase

- A purchase has a unique ID
- A purchase involves one listing
- A purchase is made by one account

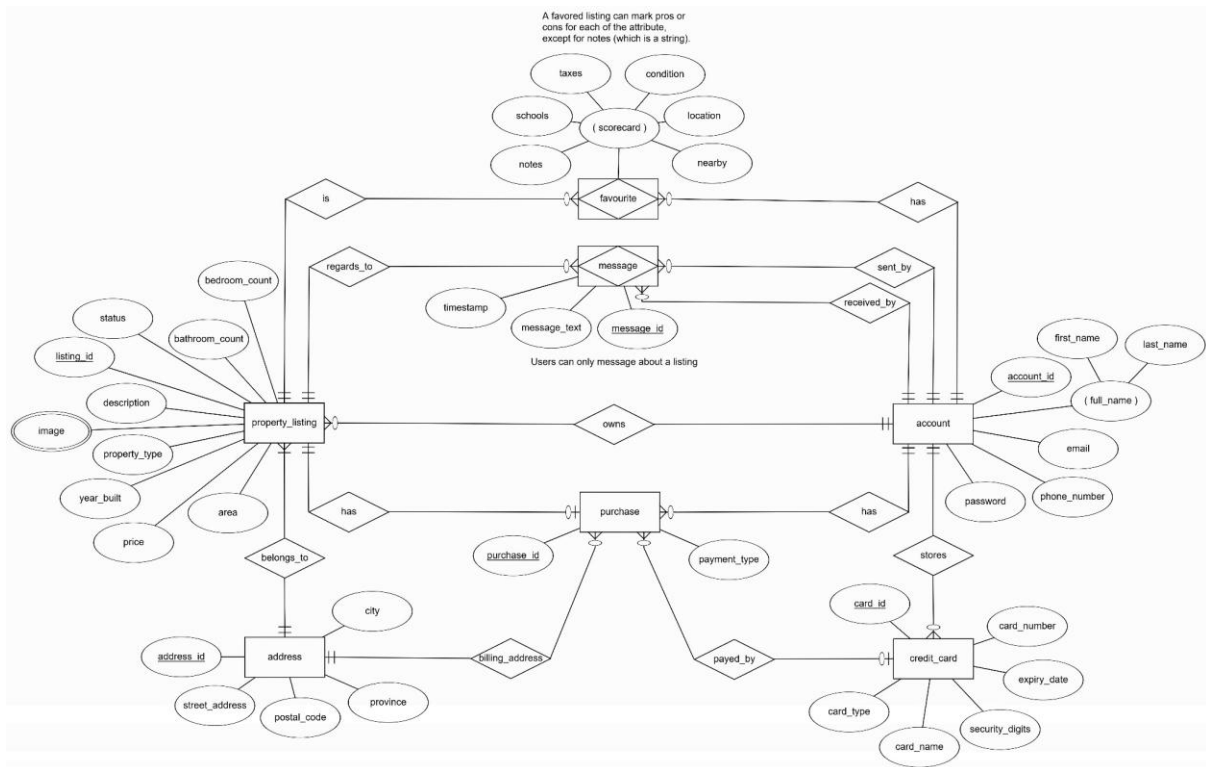
### Credit Card

- A credit card is given a unique ID (which is not its card number)
- A credit card is stored on one account
- A credit card can be used to pay 0 or more purchases

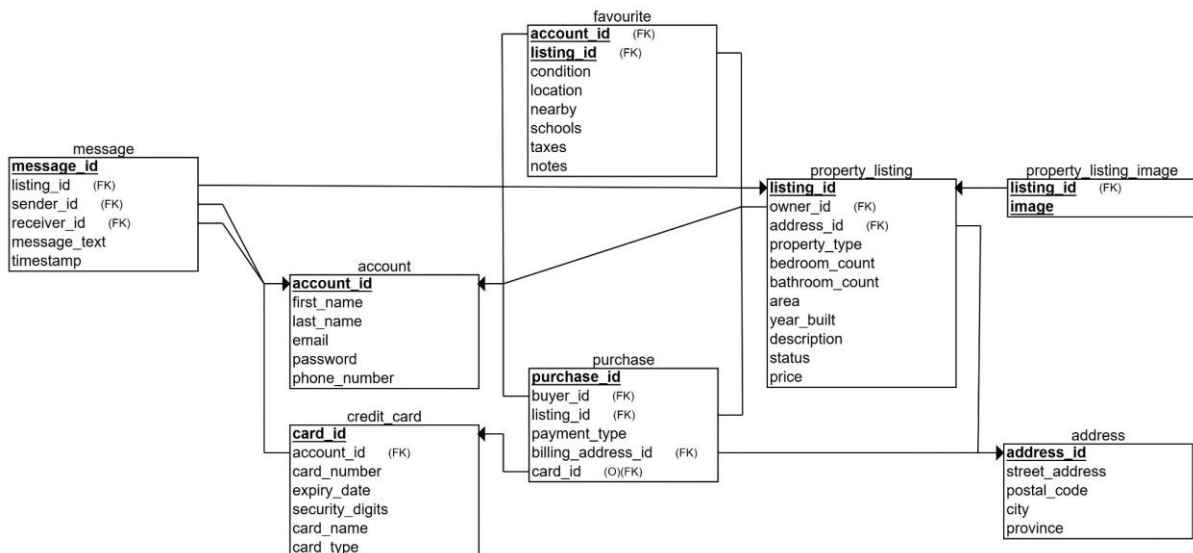
### Address

- An address has one unique ID
- An address belongs to one or more listings
- An address is the billing address of 0 or more purchases

## The E-R Data Model



## Logical Design of the Relational Database



## 3. Database design and implementation

### Table Design and Data Integrity Control

**account** Table SQL create statement:

```
CREATE TABLE account(
    account_id INT PRIMARY KEY,
    first_name VARCHAR(25),
```

```
last_name VARCHAR(25),  
email VARCHAR(50),  
password VARCHAR(50),  
phone_number char(12)  
);
```

Explanation:

**account\_id:** The primary key of this table. It is of the integer type and must be manually specified when creating a new account.

**first\_name:** A varchar attribute (string) with a maximum length of 25 characters to store the account holder's first name.

**last\_name:** A varchar attribute (string) with a maximum length of 25 characters to store the account holder's last name.

**email:** A varchar attribute (string) with a maximum length of 50 characters to store the account holder's email address.

**password:** A varchar attribute (string) with a maximum length of 50 characters to store the account password in plaintext or hash format.

**phone\_number:** A char attribute with a fixed length of 12 characters to store the account holder's phone number in a standard format.

**address Table SQL**

create statement:

```
CREATE TABLE address(  
    address_id INT PRIMARY KEY,  
    street_address VARCHAR(50),  
    postal_code CHAR(6),  
    city VARCHAR(20),  
    province CHAR(2)  
);
```

Explanation:

**address\_id:** The primary key of this table. It is of the integer type and must be manually specified when creating a new address.

**street\_address:** A varchar attribute (string) with a maximum length of 50 characters to store the street portion of the address. **postal\_code:** A char attribute with a fixed length of 6 characters to store postal codes.

**city:** A varchar attribute (string) with a maximum length of 20 characters to store the city name.

**province:** A char attribute with a fixed length of 2 characters to store the province abbreviation (e.g., ON).

### **credit\_card** Table

SQL create statement:

```
CREATE TABLE credit_card(  
    card_id INT PRIMARY KEY,  
    account_id INT,  
    card_number NUMERIC(16),  
    expiry_date DATE,  
    security_digits NUMERIC(3),  
    card_name VARCHAR(50),  
    card_type VARCHAR(20),  
    CONSTRAINT fk_credit_card_account_id FOREIGN KEY(account_id) REFERENCES  
account(account_id) ON DELETE CASCADE  
);
```

Explanation:

**card\_id:** The primary key of this table. It is of the integer type and must be manually specified.

**account\_id:** An integer type that is a foreign key linking to the account table. I have set the ON DELETE action to be CASCADE so that if an account is deleted, all associated credit cards are also deleted automatically. **card\_number:** A numeric attribute with 16 digits to store the credit card number. **expiry\_date:** A date type to store the expiry date of the credit card.

**security\_digits:** A numeric attribute with up to 3 digits to store the card's CVV/security code.

**card\_name:** A varchar attribute (string) with a maximum length of 50 characters to store the name printed on the card.

**card\_type:** A varchar attribute (string) with a maximum length of 20 characters to identify the type of card (e.g., Visa, Mastercard).

**property\_listing** Table SQL

create statement:

```
CREATE TABLE property_listing(  
    listing_id INT PRIMARY KEY,  
    owner_id INT,  
    address_id INT,  
    property_type VARCHAR(25),  
    bedroom_count NUMERIC(2),  
    bathroom_count NUMERIC(2),  
    area NUMERIC(5),  
    year_built NUMERIC(4),  
    description VARCHAR(200),  
    status VARCHAR(20),  
    price NUMERIC(15,2),  
    CONSTRAINT fk_property_owner_id FOREIGN KEY(owner_id) REFERENCES  
account(account_id) ON DELETE CASCADE,  
    CONSTRAINT fk_property_address_id FOREIGN KEY(address_id) REFERENCES  
address(address_id) ON DELETE CASCADE  
);
```

Explanation:

**listing\_id:** The primary key of this table. It is of the integer type and must be manually specified.

**owner\_id:** An integer type that is a foreign key linking to the account table. I have set the ON DELETE action to be CASCADE so that if an account is deleted, their listings are also deleted.

**address\_id:** An integer type that is a foreign key linking to the address table. I have set the ON DELETE action to be CASCADE so that deleting an address also deletes associated listings.

**property\_type:** A varchar attribute (string) with a maximum length of 25 characters to store the type of property (e.g., Condo, House).

**bedroom\_count:** A numeric attribute with a maximum of 2 digits to store the number of bedrooms.

**bathroom\_count:** A numeric attribute with a maximum of 2 digits to store the number of bathrooms.

**area:** A numeric attribute with up to 5 digits to store the area of the property in square feet.

**year\_built:** A numeric attribute with 4 digits to store the construction year of the property.

**description:** A varchar attribute (string) with a maximum length of 200 characters to describe the property.

**status:** A varchar attribute (string) with a maximum length of 20 characters to indicate if the listing is available, sold, etc.

**price:** A numeric attribute with 2 decimal points and a maximum value of 999,999,999,999.99 to store the price of the property.

**property\_listing\_image Table SQL**

create statement:

```
CREATE TABLE property_listing_image(  
    listing_id INT,  
    image VARCHAR(200),  
    CONSTRAINT fk_property_image_listing_id FOREIGN KEY(listing_id) REFERENCES  
property_listing(listing_id) ON DELETE CASCADE,  
);
```

### Explanation:

**listing\_id:** An integer type that is a foreign key linking to the property\_listing table. I have set the ON DELETE action to be CASCADE so that if a listing is deleted, its associated images are also deleted.

**image:** A varchar attribute (string) with a maximum length of 200 characters to store the URL of the property image.

### **favourite Table SQL**

#### create statement:

```
CREATE TABLE favourite(  
    account_id INT,  
    listing_id INT,  
    condition NUMERIC(1),  
    location NUMERIC(1),  
    nearby NUMERIC(1),  
    schools NUMERIC(1),  
    taxes NUMERIC(1),  
    notes VARCHAR(200),  
    CONSTRAINT fk_favourites_account_id FOREIGN KEY(account_id) REFERENCES  
account(account_id) ON DELETE CASCADE,  
    CONSTRAINT fk_favourites_listing_id FOREIGN KEY(listing_id) REFERENCES  
property_listing(listing_id) ON DELETE NO ACTION  
);
```

### Explanation:

**account\_id:** An integer type that is a foreign key linking to the account table. I have set the ON DELETE action to be CASCADE so that all favourites are deleted if the account is deleted.

**listing\_id:** An integer type that is a foreign key linking to the property\_listing table. I have set the ON DELETE action to be NO ACTION so that a listing cannot be deleted if it is marked as a favourite by any user.

**condition:** A numeric attribute with a maximum of 1 digit to store whether the condition is a “pro” or “con” for the listing (e.g. 0/1).

**location:** A numeric attribute with a maximum of 1 digit to store whether the location is a “pro” or “con” for the listing (e.g. 0/1).

**nearby:** A numeric attribute with a maximum of 1 digit to store whether the listing is in a region with lots of important places nearby (e.g. 0/1).

**schools:** A numeric attribute with a maximum of 1 digit to store school rating proximity as a “pro” or a “con” (e.g. 0/1).

**taxes:** A numeric attribute with a maximum of 1 digit to store the user’s opinion on taxes for the property as a “pro” or a “con” (e.g. 0/1).

**notes:** A varchar attribute (string) with a maximum length of 200 characters to store any additional notes or thoughts about the property.

### **message** Table SQL

#### create statement:

```
CREATE TABLE message(  
    message_id INT PRIMARY KEY,  
    listing_id INT,  
    sender_id INT,  
    receiver_id INT,  
    message_text VARCHAR(1000),  
    timestamp DATE,  
    CONSTRAINT fk_message_listing_id FOREIGN KEY(listing_id) REFERENCES  
property_listing(listing_id) ON DELETE NO ACTION,  
    CONSTRAINT fk_message_sender_id FOREIGN KEY(sender_id) REFERENCES  
account(account_id) ON DELETE NO ACTION,  
    CONSTRAINT fk_message_receiver_id FOREIGN KEY(receiver_id) REFERENCES  
account(account_id) ON DELETE NO ACTION  
);
```

#### Explanation:

**message\_id:** The primary key of this table. It is of the integer type and must be manually specified.

**listing\_id:** An integer type that is a foreign key linking to the property\_listing table. I have set the ON DELETE action to be NO ACTION so that listings cannot be deleted while they have associated messages.

**sender\_id:** An integer type that is a foreign key linking to the account table. I have set the ON DELETE action to be NO ACTION so that an account cannot be deleted while they have associated messages.

**receiver\_id:** An integer type that is a foreign key linking to the account table. Same logic as for the sender\_id with the ON DELETE action set as NO ACTION so that an account cannot be deleted while they have associated messages.

**message\_text:** A varchar attribute (string) with a maximum length of 1000 characters to store the content of the message.

**timestamp:** A date type to store when the message was sent.  
create statement:

**purchase** Table SQL



```

CREATE TABLE purchase(
    purchase_id INT PRIMARY KEY,
    buyer_id INT,
    listing_id INT,
    payment_type VARCHAR(20),
    billing_address_id INT,
    card_id INT NULL,
    CONSTRAINT fk_purchase_buyer_id FOREIGN KEY(buyer_id) REFERENCES
account(account_id) ON DELETE NO ACTION,
    CONSTRAINT fk_purchase_listing_id FOREIGN KEY(listing_id) REFERENCES
property_listing(listing_id) ON DELETE NO ACTION,
    CONSTRAINT fk_purchase_card_id FOREIGN KEY(card_id) REFERENCES
credit_card(card_id) ON DELETE SET NULL,
);

```

#### Explanation:

**purchase\_id:** The primary key of this table. It is of the integer type and must be manually specified.

**buyer\_id:** An integer type that is a foreign key linking to the account table. I have set the ON DELETE action to be NO ACTION to prevent deletion of an account with purchase records.

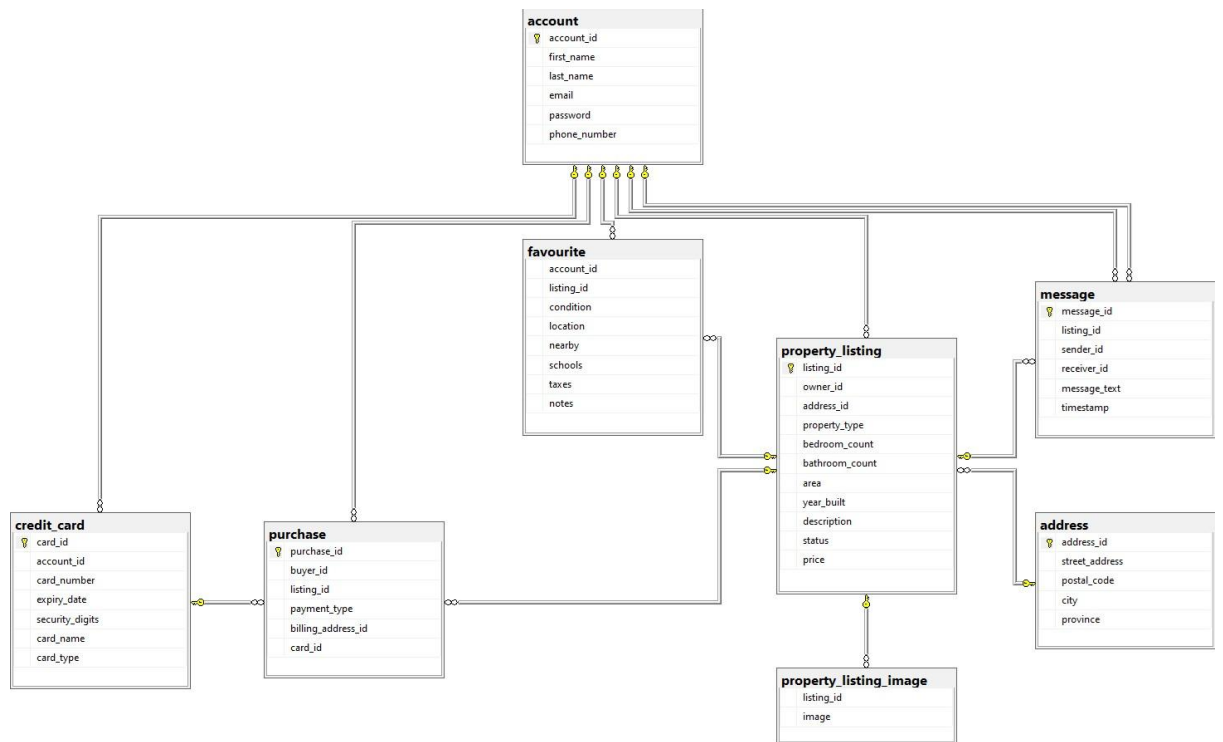
**listing\_id:** An integer type that is a foreign key linking to the property\_listing table. I have set the ON DELETE action to be NO ACTION to prevent deletion of listings with purchase records.

**payment\_type:** A varchar attribute (string) with a maximum length of 20 characters to indicate the method of payment (e.g., credit, PayPal).

**billing\_address\_id:** An integer type that is meant to reference an address, although this is not explicitly declared as a foreign key in the schema.

**card\_id:** An integer type that is a foreign key referencing the credit\_card table. I have set the ON DELETE action to be SET NULL so that if a card is deleted, the purchase record remains but the card\_id becomes null.

## Design Relational Diagram



### Sample Data for all the Tables

**account** Table Sample Data: 7 records given

<u>account_id</u>	first_name	last_name	email	password	phone_number
1	Jordan	Yan	jy@gmail.com	pw123	437-911-4258
2	Ronald	Kirk	ronaldk@outlook.com	123123!	647-184-0001
3	Trung	Nguyen	t@nguyen.com	54321pass	416-818-4242
4	Soham	Sood	sohs3@hotmail.com	128250Jk!	416-3450-232
5	Zenith	Saw	zenith@saw.ca	password321	647-234-0389
6	John	Doe	johndoe@email.com	ThisIsPassword	555-123-4567
7	Bob	Kim	bobk93@my.yorku.ca	bobkim951	437-010-9541
8	Owen	Miller	omiller@gmail.com	pw2354kj	555-456-4567

#### SQL Insert Statements:

```
INSERT INTO account (account_id, first_name, last_name, email, password, phone_number)
VALUES

(1, 'Jordan', 'Yan', 'jy@gmail.com', 'pw123', '437-911-4258'),

(2, 'Ronald', 'Kirk', 'ronaldk@outlook.com', '1231231', '647-184-0001'),

(3, 'Trung', 'Nguyen', 't@nguyen.com', '54321pass', '416-818-4242'),

(4, 'Soham', 'Sood', 'sohs3@hotmail.com', '128250Jkj1', '416-345-0232'),

(5, 'Zenith', 'Saw', 'zenith@saw.ca', 'password321', '647-234-0389'),

(6, 'John', 'Doe', 'johndoe@email.com', 'ThisIsPassword', '555-123-4567'),

(7, 'Bob', 'Kim', 'bobk93@my.yorku.ca', 'bobkim951', '437-010-9541'),

(8, 'Owen', 'Miller', 'omiller@gmail.com', 'pw2354jkj', '555-456-4567');
```

**address** Table Sample Data:  
7 records given

<u>address_id</u>	street_address	postal_code	city	province
1	500 Yonge St	M3M4J7	Toronto	ON

2	1000 Yonge St	M4N9K0	Toronto	ON
3	123 Main St	M5V2H1	Toronto	ON
4	555 Maple Dr	K1P5J7	Ottawa	ON
5	103 Oakwood Road	N2N4K2	Oakville	ON
6	93 Oakwood Dr	L5M4K9	Mississauga	ON
7	123 Main St	P4N0H8	Hamilton	ON

#### SQL Insert Statements:

```
INSERT INTO address (address_id, street_address, postal_code, city, province) VALUES
(1, '500 Yonge St', 'M3M417', 'Toronto', 'ON'),
(2, '1000 Yonge St', 'M4N9K0', 'Toronto', 'ON'),
(3, '123 Main St', 'M5V2H1', 'Toronto', 'ON'),
(4, '555 Maple Dr', 'K1P517', 'Ottawa', 'ON'),
(5, '103 Oakwood Road', 'N2N4K2', 'Oakville', 'ON'),
(6, '93 Oakwood Dr', 'L5M4K9', 'Mississauga', 'ON'),
(7, '123 Main St', 'P4N0H8', 'Hamilton', 'ON'),
(8, '789 Rainbow Rd', 'S1T2U3', 'Markham', 'ON'),
(9, '42 Unicorn St', 'V4W5X6', 'Bradford', 'ON'),
(10, '456 Dragon Way', 'Y7Z8A9', 'Newmarket', 'ON'),
(11, '93 Wizard St', 'B1C2D3', 'Hamilton', 'ON'),
(12, '123 Phoenix Ave', 'E4F5G6', 'Richmond Hill', 'ON'),
(13, '246 Bloor Rd', 'H7I8J9', 'Barrie', 'ON'),
(14, '78 Doris Ave', 'K1L2M3', 'Brampton', 'ON'),
(15, '10 Griffin St', 'N4O5P6', 'Hamilton', 'ON');
```

#### **credit\_card** Table

Note that the card\_number should be of type char(16) because it contains so many characters

Sample Data: 5 records given

<u>card_id</u>	account_id	card_number	expiry_date	security_digits	card_name	card_type
----------------	------------	-------------	-------------	-----------------	-----------	-----------

1	1	4532875498761234	2026-08-31	123	Jordan Yan	Visa
2	2	4111111111111111	2027-01-31	456	Ronald Kirk	MasterCard
3	3	6011111111111117	2026-04-05	111	Trung Hieu	Visa
4	4	5500000000000004	2026-04-06	913	Soham Sood	Amex
5	5	5500000000001678	2027-11-28	777	Zenith Saw	MasterCard

SQL Insert Statements:

```
INSERT INTO credit_card (card_id, account_id, card_number, expiry_date, security_digits,
card_name, card_type) VALUES

(1, 1, '4532875498761234', '2026-08-31', '123', 'Jordan Yan', 'Visa'),

(2, 2, '4111111111111111', '2027-01-31', '456', 'Ronald Kirk', 'MasterCard'),

(3, 3, '6011111111111117', '2026-04-05', '111', 'Trung Hieu', 'Visa'),

(4, 4, '5500000000000004', '2026-04-06', '913', 'Soham Sood', 'Amex'),

(5, 5, '5500000000001678', '2027-11-28', '777', 'Zenith Saw', 'MasterCard'),

(6, 6, '378282246310005', '2025-12-31', '234', 'John Doe', 'Amex'),

(7, 7, '5555555555554444', '2026-06-30', '567', 'Bob Kim', 'MasterCard'),

(8, 8, '5105105105105100', '2027-03-31', '890', 'Owen Miller', 'MasterCard');
```

**property\_listing** Table

Sample Data: 5 records given

<u>listing_id</u>	owner_id	address_id	property_type	bedroom_count	bathroom_count	area	year_built	description	status	price
1	1	3	Condo	3	2	1200	1995	Condo with 2 bathrooms	For Rent	599000

2	1	2	Condo	1	1	750	2010	Small condo in urban area	For Sale	350000
3	2	15	House	2	1	1800	2003	Near the park	Sold	850000
4	3	1	Townhouse	3	3	2100	2011	Large townhouse in the suburbs	Pending	425000
5	1	10	House	4	2	1500	2000	Updated family home	Pending	1200000

SQL Insert Statements:

```

INSERT INTO property_listing (listing_id, owner_id, address_id, property_type, bedroom_count,
bathroom_count, area, year_built, description, status, price) VALUES

(1, 1, 3, 'Condo', 3, 2, 1200, 1995, 'Condo with 2 bathrooms', 'For Rent', 599000),
(2, 1, 2, 'Condo', 1, 1, 750, 2010, 'Small condo in urban area', 'For Sale', 350000),
(3, 2, 15, 'House', 2, 1, 1800, 2003, 'Large house with 1 bathroom', 'Sold', 850000),
(4, 3, 1, 'Townhouse', 3, 3, 2100, 2011, 'Large townhouse in the suburbs', 'Pending', 425000),
(5, 1, 10, 'House', 4, 2, 1500, 2000, 'Updated family home', 'Pending', 1200000),
(6, 2, 7, 'House', 5, 4, 2700, 1987, 'Older house in downtown', 'For Sale', 375000),
(7, 8, 9, 'House', 2, 2, 1500, 2024, 'Cozy new-built home', 'Sold', 675000),
(8, 4, 4, 'Townhouse', 3, 3, 3000, 2007, 'Large, spacious townhouse', 'Sold', 1500000),
(9, 5, 5, 'Condo', 2, 1, 2000, 2007, 'Pet-friendly building', 'For Sale', 150000),
(10, 2, 6, 'House', 5, 5, 4500, 2001, 'Quiet neighbourhood', 'For Rent', 3250000),
(11, 6, 8, 'Condo', 2, 2, 1600, 2015, 'Luxury condo with lake view', 'For Sale', 775000),
(12, 7, 11, 'House', 3, 3, 2200, 2018, 'Modern interior design home', 'For Sale', 925000),
(13, 3, 12, 'Condo', 1, 1, 800, 2020, 'Downtown unit', 'For Rent', 425000),
(14, 4, 13, 'House', 5, 4, 3200, 2012, 'Cozy home', 'Pending', 1850000),
(15, 5, 14, 'Townhouse', 3, 2, 1400, 2005, 'Near the park', 'Sold', 525000);

```

property\_listing\_image Table

Sample Data: 5 records given

<u>listing_id</u>	<u>image</u>
-------------------	--------------

3	https://ph.cincmedia.com/ontrebvowapi/7142fa973ad6f7ac599fd669e03a1954.jpg?width=1024
3	https://ph.cincmedia.com/ontrebvowapi/72e4c89042f0173885899391000619d9.jpg?width=512
14	https://ph.cincmedia.com/ontrebidx/488ad1d873888e38713b52f919fc8c41.jpg?width=1024
14	https://ph.cincmedia.com/ontrebidx/a9796b4880129348e635b54742570638.jpg?width=512
14	https://ph.cincmedia.com/ontrebidx/c428b8136917ce5504d119f2098aac21.jpg?width=512
12	https://ph.cincmedia.com/ontrebvowapi/c7cf5202b773b93b8b383a82189a1567.jpg?width=1024
12	https://ph.cincmedia.com/ontrebvowapi/c27a82168f483aceca527e47ed188a43.jpg?width=512
12	https://ph.cincmedia.com/ontrebvowapi/82146926d034ef50ad54bef7b2d76f34.jpg?width=512

#### SQL Insert Statements:

```
INSERT INTO property_listing_image (listing_id, image) VALUES
(3, 'https://ph.cinemediia.com/ontrebvowapi/7142fa973ad6f7ac599fd669e03a1954.jpg?width=1024'),
(3, 'https://ph.cinemediia.com/ontrebvowapi/72e4c89042b0173885899391000619d9.jpg?width=512'),
(14, 'https://ph.cinemediia.com/ontrebidx/488ad1d873888e38713b52f919fc8c41.jpg?width=1024'),
(14, 'https://ph.cinemediia.com/ontrebidx/a979604880129348e635b54742570638.jpg?width=512'),
(14, 'https://ph.cinemediia.com/ontrebidx/c428k8136917ce5504d119f2098aac21.jpg?width=512'),
(12, 'https://ph.cinemediia.com/ontrebvowapi/c7cf5202b773b93b853a82189a1567.jpg?width=1024'),
(12, 'https://ph.cinemediia.com/ontrebvowapi/c27a82168f483aceea527e47ed188a43.jpg?width=512'),
(12, 'https://ph.cinemediia.com/ontrebvowapi/82146926d034ef50ad54bef7b2d76f34.jpg?width=512');
```

#### favourite Table Sample Data:

4 records given

<u>account_id</u>	<u>listing_id</u>	condition	location	nearby	schools	taxes	notes
-------------------	-------------------	-----------	----------	--------	---------	-------	-------

2	1	Good	Downtown	Transit	Yes	Low	Good for a small family
1	3	Moderate	Quiet Area	Park	No	Low	Very spacious
1	15	Poor	Rural Area	Park	Yes	High	Good for medium-sized families
4	12	Excellent	Lakeside Hamilton	Lake	No	Medium	Luxury condo near the lake

SQL Insert Statements:

```
INSERT INTO favourite (account_id, listing_id, condition, location, nearby,
schools, taxes, notes) VALUES
(2, 1, 1, 1, 1, 1, 1, 'Good for a small family'),
(1, 3, 0, 1, 0, 0, 1, 'Very spacious'),
(1, 15, 0, 0, 1, 1, 0, 'Good for medium-sized families'),
(4, 12, 1, 1, 1, 0, 1, 'Luxury condo near the lake'),
(3, 5, 1, 1, 0, 1, 1, 'Great neighborhood with top-rated schools'),
(5, 7, 1, 0, 0, 0, 0, 'Modern condo in the city center'),
(6, 9, 0, 1, 1, 1, 1, 'Beautiful views of the harbor');
```

message Table

Sample Data: 7 records given

message_id	listing_id	sender_id	receiver_id	message_text	timestamp
------------	------------	-----------	-------------	--------------	-----------



1	1	2	6	Hello, I would like to discuss this property. Is it possible to get a lower price on it?	2023-05-15 10:30:45
2	1	6	2	We have some flexibility. What were you thinking	2023-07-04 12:01:39
3	1	2	6	How safe is the neighbourhood of this house?	2023-07-04 16:30:01
4	1	6	2	It's in a pretty safe area. That's why you'll likely have to pay a premium because of this.	2024-01-03 01:59:16
5	1	2	6	Anything over 2 million is too much for me.	2024-01-10 02:29:47
6	2	1	7	Can I schedule a viewing for this condo?	2024-11-30 13:30:45
7	2	7	1	Available tomorrow at 2 pm or Friday morning.	2024-12-21 23:04:19

#### SQL Insert Statements:

```
INSERT INTO message (message_id, listing_id, sender_id, receiver_id, message_text, timestamp)
VALUES

(1, 1, 2, 6, 'Hello, I would like to discuss this property. Is it possible to get a lower price on it?', '2023-05-15 10:30:45'),

(2, 1, 6, 2, 'We have some flexibility. What were you thinking', '2023-07-04 12:01:39'),

(3, 1, 2, 6, 'How safe is the neighbourhood of this house?', '2023-07-04 16:30:01'),

(4, 1, 6, 2, 'It's in a pretty safe area. That's why you'll likely have to pay a premium because of this.', '2024-01-03 01:59:16'),

(5, 1, 2, 6, 'Anything over 2 million is too much for me.', '2024-01-10 02:29:47'),

(6, 2, 7, 1, 'Can I schedule a viewing for this condo?', '2024-11-30 13:30:45'),

(7, 2, 1, 7, 'Available tomorrow at 2 pm or Friday morning.', '2024-12-21 23:04:19'),

(8, 2, 7, 1, 'Are pets allowed in this townhouse complex?', '2025-05-23 13:20:37'),

(9, 2, 1, 7, 'Yes, you are allowed up to 2 pets.', '2025-05-23 14:15:41'),

(10, 5, 3, 7, 'Hi. I would like to meet about this property', '2025-05-24 01:00:52'),

(11, 5, 7, 3, 'I'm available next Thursday all day. Let me know about your availability.', '2025-05-24 08:40:02');
```

**purchase** Table Sample Data:

4 records given

<u>purchase_id</u>	buyer_id	listing_id	payment_type	billing_address_id	card_id
1	1	3	Credit	15	1
2	1	7	Credit	9	1
3	3	8	PayPal	4	
4	4	15	Credit	14	4

**SQL Insert Statements:**

```
INSERT INTO purchase (purchase_id, buyer_id, listing_id, payment_type,
billing_address_id, card_id) VALUES

(1, 1, 3, 'Credit', 15, 1),

(2, 1, 7, 'Credit', 9, 1),

(3, 3, 8, 'PayPal', 4, NULL),

(4, 4, 15, 'Credit', 14, 4);
```

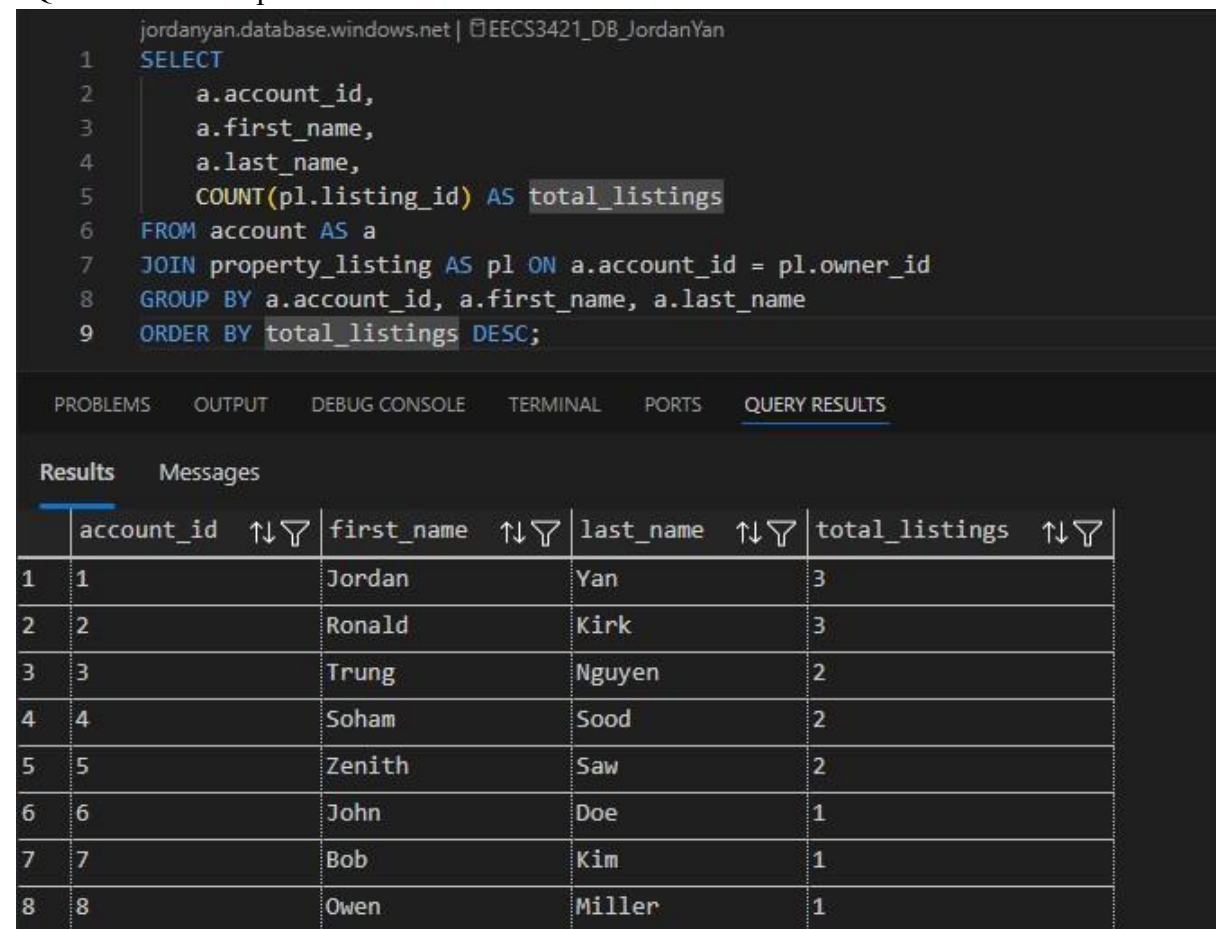
**Sample SQL Queries to Support Applications and Reports for Management Use**

## Query 1: Number of listings per owner SQL

Code:

```
SELECT
    a.account_id,
    a.first_name,
    a.last_name,
    COUNT(pl.listing_id) AS total_listings
FROM account AS a
JOIN property_listing AS pl ON a.account_id = pl.owner_id
GROUP BY a.account_id, a.first_name, a.last_name
ORDER BY total_listings DESC;
```

SQL Code and Output:



The screenshot shows a SQL IDE interface. At the top, the connection string is 'jordanyan.database.windows.net | EECS3421\_DB\_JordanYan'. The SQL code is entered in the editor, and the 'QUERY RESULTS' tab is selected. Below the code, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'QUERY RESULTS'. The 'QUERY RESULTS' tab shows a table with 5 columns: 'account\_id', 'first\_name', 'last\_name', and 'total\_listings'. The table contains 8 rows of data, ordered by 'total\_listings' in descending order.

	account_id	first_name	last_name	total_listings
1	1	Jordan	Yan	3
2	2	Ronald	Kirk	3
3	3	Trung	Nguyen	2
4	4	Soham	Sood	2
5	5	Zenith	Saw	2
6	6	John	Doe	1
7	7	Bob	Kim	1
8	8	Owen	Miller	1

### Explanation for Management:

This query shows how many properties each user/account has listed. It helps identify the most active sellers in the system, and we ordered it by whoever has the most total listings first (i.e. descending order).

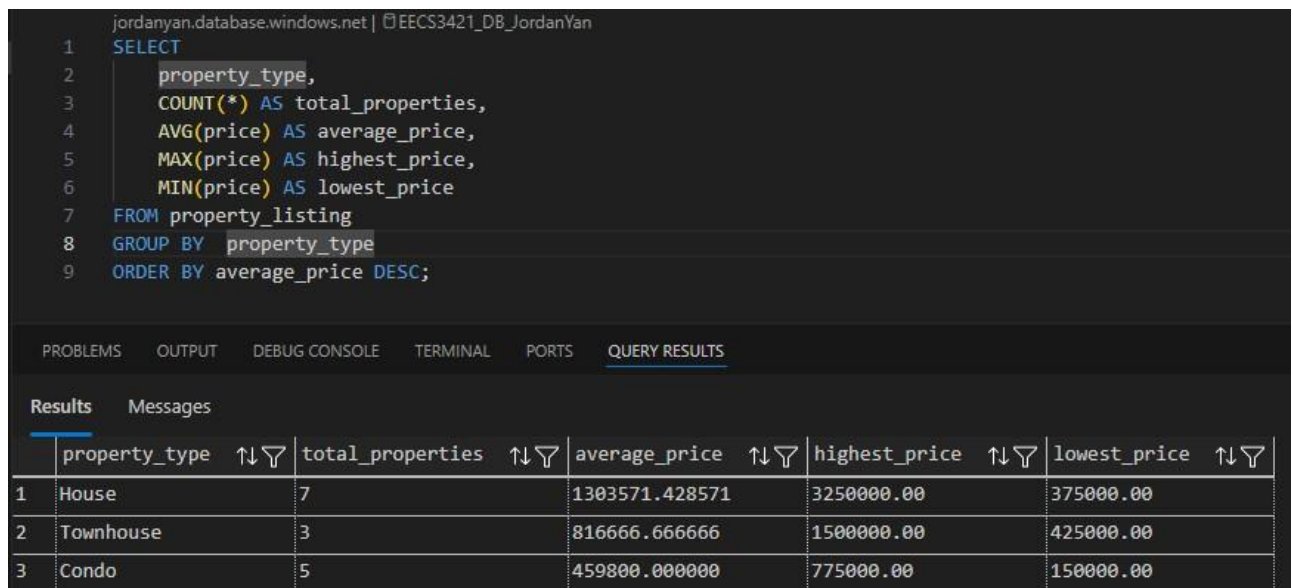
## Query 2:

SQL Code:

### Measurements for each property type

```
SELECT
    property_type,
    COUNT(*) AS total_properties,
    AVG(price) AS average_price,
    MAX(price) AS highest_price,
    MIN(price) AS lowest_price
FROM property_listing
GROUP BY property_type
ORDER BY average_price DESC;
```

SQL Code and Output:



The screenshot shows a database IDE with the following components:

- SQL Editor:** Contains the SQL query for Query 2.
- Query Results Tab:** Displays the results of the query in a table format.

**SQL Code:**

```
1 SELECT
2     property_type,
3     COUNT(*) AS total_properties,
4     AVG(price) AS average_price,
5     MAX(price) AS highest_price,
6     MIN(price) AS lowest_price
7 FROM property_listing
8 GROUP BY property_type
9 ORDER BY average_price DESC;
```

**Query Results:**

	property_type	total_properties	average_price	highest_price	lowest_price
1	House	7	1303571.428571	3250000.00	375000.00
2	Townhouse	3	816666.666666	1500000.00	425000.00
3	Condo	5	459800.000000	775000.00	150000.00

### Explanation for Management:

This query shows basic measurements for each property type, including total listings, average price, maximum price and minimum price for each property type. We ordered it by the average cost of each property type in descending order. This query could be helpful to the company to see what the appropriate pricing for certain property types is.

### List of transactions with buyer and seller information

```
SELECT
    p.purchase_id,
    buyer.first_name AS buyer_first_name,
```

### Query 3:

#### SQL Code:

```
        seller.first_name AS seller_first_name,  
pl.price,  
  
        pl.property_type,  
        p.payment_type  
FROM purchase AS p  
JOIN account AS buyer ON p.buyer_id = buyer.account_id  
JOIN property_listing AS pl ON p.listing_id = pl.listing_id  
JOIN account AS seller ON pl.owner_id = seller.account_id;
```

#### SQL Code and Output:

```
jordanyan.database.windows.net | EECS3421_DB_JordanYan  
1  SELECT  
2      p.purchase_id,  
3      buyer.first_name AS buyer_first_name,  
4      seller.first_name AS seller_first_name,  
5      pl.price,  
6      pl.property_type,  
7      p.payment_type  
8  FROM purchase AS p  
9  JOIN account AS buyer ON p.buyer_id = buyer.account_id  
10 JOIN property_listing AS pl ON p.listing_id = pl.listing_id  
11 JOIN account AS seller ON pl.owner_id = seller.account_id;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

Results Messages

	purchase_id	buyer_first_name	seller_first_name	price	property_type	payment_type
1	1	Jordan	Ronald	850000.00	House	Credit
2	2	Jordan	Owen	675000.00	House	Credit
3	3	Trung	Soham	1500000.00	Townhouse	PayPal
4	4	Soham	Zenith	525000.00	Townhouse	Credit

#### Explanation for Management:

This query allows us to view a transaction in more detail. Specifically, we can see all parties involved in a transaction, and we get the first names of the home buyer and home seller.

Additionally, we have details about the property in the transaction, such as price and property type, which may be useful information.

#### **Total messages sent for each listing**

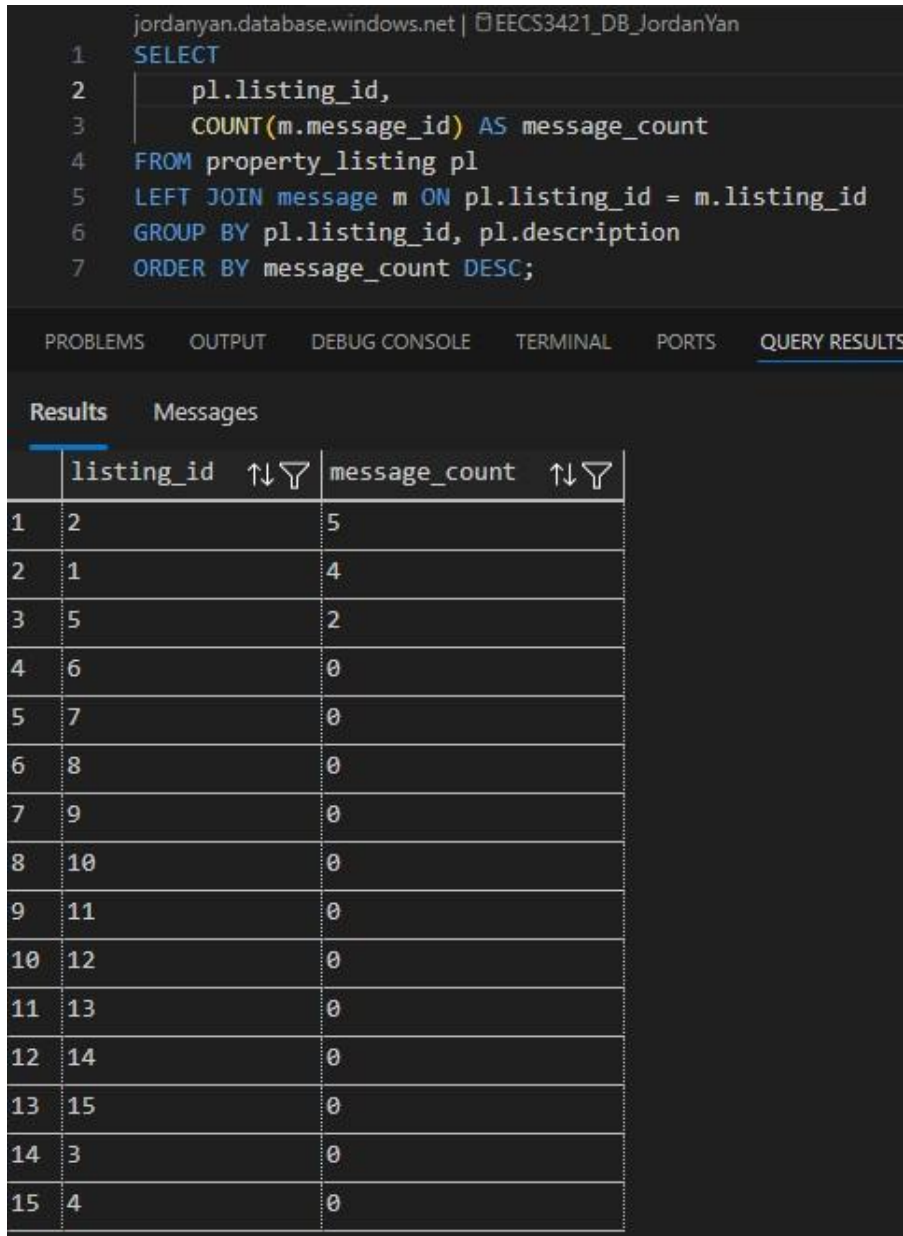
```
SELECT  
  
    pl.listing_id,  
    COUNT(m.message_id) AS message_count  
FROM property_listing AS pl  
LEFT JOIN message AS m ON pl.listing_id = m.listing_id
```

#### Query 4:

SQL Code:

```
GROUP BY pl.listing_id, pl.description  
ORDER BY message_count DESC;
```

SQL Code and Output:



The screenshot shows a SQL IDE interface. At the top, the connection string is 'jordanyan.database.windows.net | EECS3421\_DB\_JordanYan'. The query editor contains the following SQL code:

```
1 SELECT  
2   pl.listing_id,  
3   COUNT(m.message_id) AS message_count  
4 FROM property_listing pl  
5 LEFT JOIN message m ON pl.listing_id = m.listing_id  
6 GROUP BY pl.listing_id, pl.description  
7 ORDER BY message_count DESC;
```

Below the query editor, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'QUERY RESULTS'. The 'QUERY RESULTS' tab is selected, showing a table with two columns: 'listing\_id' and 'message\_count'. The table is sorted by 'message\_count' in descending order.

	listing_id	message_count
1	2	5
2	1	4
3	5	2
4	6	0
5	7	0
6	8	0
7	9	0
8	10	0
9	11	0
10	12	0
11	13	0
12	14	0
13	15	0
14	3	0
15	4	0

#### Explanation for Management:

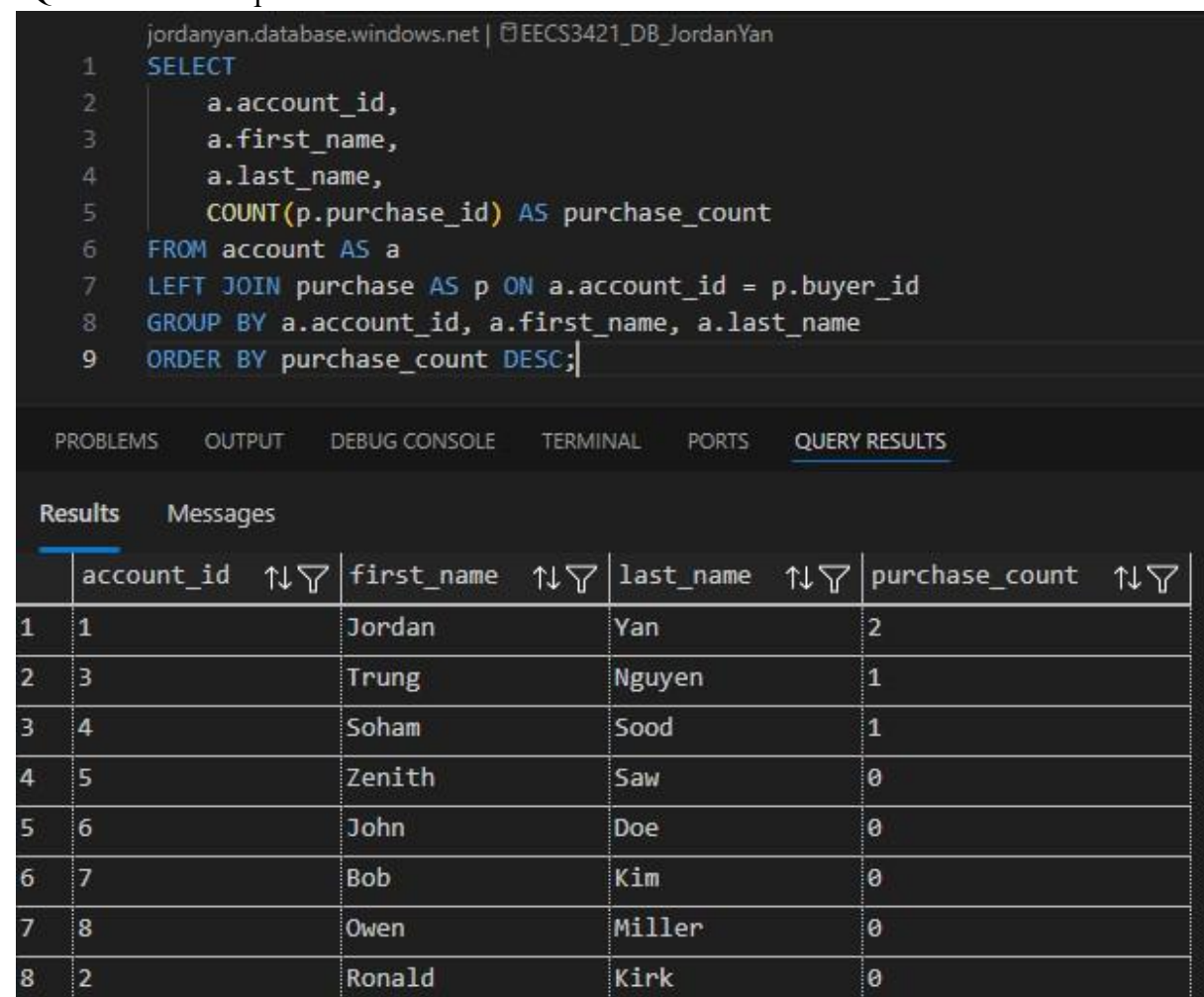
This query can be used to see which properties have the most interest in them (i.e. messages about the property) by ordering them by the total messages about them.

### Query 5: Number of properties bought by each user SQL

Code:

```
SELECT
    a.account_id,
    a.first_name,
    a.last_name,
    COUNT(p.purchase_id) AS purchase_count
FROM account AS a
LEFT JOIN purchase AS p ON a.account_id = p.buyer_id
GROUP BY a.account_id, a.first_name, a.last_name
ORDER BY purchase_count DESC;
```

SQL Code and Output:



The screenshot shows a SQL IDE interface. At the top, the connection string is 'jordanyan.database.windows.net | EECS3421\_DB\_JordanYan'. The SQL query is entered in the editor and is identical to the one in the previous block. Below the editor, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'QUERY RESULTS'. The 'QUERY RESULTS' tab is selected, showing a table with 8 rows and 5 columns: 'account\_id', 'first\_name', 'last\_name', and 'purchase\_count'. The results are ordered by 'purchase\_count' in descending order.

	account_id	first_name	last_name	purchase_count
1	1	Jordan	Yan	2
2	3	Trung	Nguyen	1
3	4	Soham	Sood	1
4	5	Zenith	Saw	0
5	6	John	Doe	0
6	7	Bob	Kim	0
7	8	Owen	Miller	0
8	2	Ronald	Kirk	0

#### Explanation for Management:

This query can be used to see which users are the most active in terms of buying properties.

Here we see the total properties bought by each user and order them by whoever has the most total properties bought first (i.e. descending order).

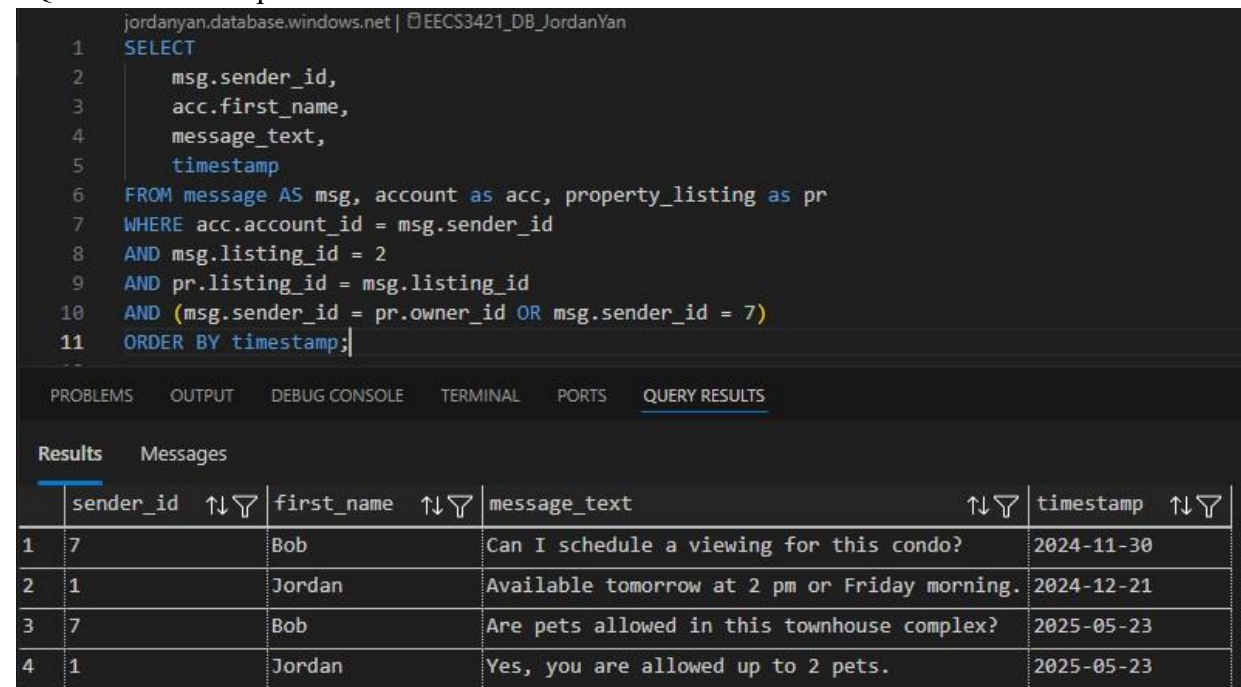


## Query 6: Messages between 2 users for a specific listing SQL

Code:

```
SELECT
    msg.sender_id,
    acc.first_name,
    message_text,
    timestamp
FROM message AS msg, account as acc, property_listing as pr
WHERE acc.account_id = msg.sender_id
AND msg.listing_id = 2
AND pr.listing_id = msg.listing_id
AND (msg.sender_id = pr.owner_id OR msg.sender_id = 7)
ORDER BY timestamp;
```

SQL Code and Output:



The screenshot shows a database IDE interface. At the top, the connection string is 'jordanyan.database.windows.net | EECS3421\_DB\_JordanYan'. The SQL query is entered in the editor and is identical to the one in the previous block. Below the editor, the 'QUERY RESULTS' tab is active, displaying a table with 4 rows and 5 columns. The columns are 'sender\_id', 'first\_name', 'message\_text', and 'timestamp'. The rows represent messages sent by user 7 (Bob) and user 1 (Jordan) regarding listing 2.

	sender_id	first_name	message_text	timestamp
1	7	Bob	Can I schedule a viewing for this condo?	2024-11-30
2	1	Jordan	Available tomorrow at 2 pm or Friday morning.	2024-12-21
3	7	Bob	Are pets allowed in this townhouse complex?	2025-05-23
4	1	Jordan	Yes, you are allowed up to 2 pets.	2025-05-23

### Explanation for the Application:

This query can be used to see the conversation between two users regarding a specific listing. It will show the name and ID of the users sending the message, as well as the time it was sent and the message itself. The numbers in the WHERE clause of the query (2 and 7 in the example) are the listing ID for which we want to see the messages and the user ID of the user interested in the listing (as the ID of the owner of the listing is known). This query can be used in the application to display their conversation.



## **4. Evaluation and conclusion**

### **The Achievement of the Project**

This project successfully implements a database system for Hamilton Area Home Listings. The organization's business objective is to provide a platform for the buying and selling of properties in the Hamilton area of Ontario. It meets the functional requirements such as user account management, listing creation, browsing and searching of listings, listing details, and a messaging system. We maintained comprehensive data modelling as we created a relational structure that accurately represents real-world real estate transactions, through the use of tables for users, properties, financials, and interactions. For instance, the `property_listing` table encapsulates all the necessary fields, including type, beds/baths, price, etc., while simultaneously maintaining the relationship between owners and addresses. Furthermore, we employed different levels of data integrity within our database design. For example, entity integrity through the assignment of primary keys (e.g. `account_id`, `listing_id`, etc.), referential integrity through the appropriate use of ON DELETE actions (e.g. cascading for credit cards, no action for purchases, etc.), and domain integrity through the assignment of proper data types (e.g. `VARCHAR(25)` for `property_type`, `NUMERIC(15,2)` for price, etc.). The ability to track which user has the most properties helps with recognizing top-performing sellers, which allows the organization to reward them. This, in turn, can encourage more listings. Likewise, it helps identify frequently active users who may benefit from special/premium services catered to their own needs. Tracking transaction details creates transparency within the sales process and also allows agents to connect with past buyers and sellers for repeat business. The ability to monitor the number of messages per listing aids in identifying properties gaining lots of attention and also reveals which properties are not doing so well and may need listing adjustments. These are some of the aspects of the database system that encapsulate the overall business objective of Hamilton Area Home Listings.

### **Further Development and Improvement**

For further development and improvement of our database design, we can consider several key refinements to strengthen it. First, data integrity can be improved by enhancing the constraints. For example, NOT NULL constraints for fields such as account first\_name and primary keys in other tables. Likewise, we should consider check constraints to uphold business rules, such as ensuring that property prices are greater than 0 (CHECK(price > 0)). Moreover, security improvements can be made for the sake of user safety. In particular, replacing plain text password storage with secure hashing in dedicated columns. Tracking a user's last login would also aid with user monitoring. To optimize the performance of our database design, indexing can be implemented. Specifically, indexing should be applied to frequently queried columns, for example, property\_type, bedroom\_count, etc. Implementing this will accelerate search performance for users. These refinements will collectively improve data quality, security, and query performance while maintaining the overall integrity of our database design.

### **The Advantages and Limitations of the Database Package Used**

For our group project, we required a database service solution that would support cross-platform development, minimize setup and maintenance overhead for each team member, and enable seamless collaboration. With team members using different operating systems and development environments, we found it essential to choose a platform that offered consistency and accessibility without the complications of managing local infrastructure. These needs guided our decision to adopt a cloud-based approach over traditional local server options.

#### The advantages of using Azure SQL Database:

1. A centralized and collaborative environment:

The ability to connect to the same database instance meant that everyone was working with the latest data, schema and changes, thus eliminating any inconsistency or miscommunication. We are also able to connect to this database from any location with an internet connection, enabling remote work and flexible schedules. Overall, this simplified the collaborative process, as sharing database scripts and applying changes became much smoother when everyone hit the same endpoint.

2. Managed Service: Less Admin Overhead

Microsoft generally handled all the underlying infrastructure (hardware, operating system, SQL Server software installation, patching, updates), allowing our group to stay focused on each task at hand, rather than worrying about confusing and time-consuming configuration issues. Azure is also highly available and reliably accessible as it features built-in mechanisms that ensure high uptime and resilience so that the database is generally available anytime we need it, even if there are underlying hardware issues in the data center. This also included a benefit of having automated backups, including point-in-time restore capabilities, which is a reliable service that ensured protection against any accidental data loss, or critical errors. The inclusion of this was a crucial safeguard, which eliminated the need for manual backup procedures, offering considerable peace of mind.

### 3. Security Features:

Azure provided access to a system of robust built-in security features that would be impractical to implement on a local setup. While undertaking this project, it was always reassuring to know that these security protocols were in place, even if our primary focus was centred on core project development. The platform offered essential components such as configurable firewalls for precise access control, automated threat detection, and advanced data encryption, without demanding any direct intervention or extensive technical configuration on our part.

### 4. Azure for Student credits pricing feature

The Azure for Students program presented a big advantage for our group, which fundamentally lowered the barrier to entry for utilizing cloud-based database services. This allowed us full accessibility to Azure's powerful features, including a fully managed SQL database entirely without upfront financial commitment. It was a seamless and risk-free onboarding experience, which allowed us to immediately begin developing. The program also acted as a built-in safeguard, as it prevented any accidental or unexpected charges should we inadvertently leave resources running or exceed the usage limit. This financial predictability allowed our team to focus purely on the technical aspects of our project, further enabling a worry-free environment.

### The disadvantages of using Azure SQL Database:

#### 1. Initial Network Configuration Complexity (Minor)

The initial network configuration presented a minor layer of complexity. As a security measure, explicitly defining firewall rules for allowed client IP addresses was necessary. However, for team members with dynamic IP addresses, this sometimes required re-updating the firewall settings, which was a small but recurring inconvenience.

## 2. Performance Tiers and Resource Governance

Our reliance on the basic or serverless performance tiers chosen to align with free usage limits meant accepting certain performance constraints and resource governance. While this was sufficient for our project's needs, more demanding applications would likely require scaling up to higher cost tiers to avoid potential performance problems.

Furthermore, despite the availability of generous free tiers and student credits, diligent cost management required active monitoring of resource consumption within the Azure Portal. Failing to properly manage or terminate resources could have led to the suspension of active services once credits were exhausted. Therefore, adhering to our free limits was crucial for operation throughout the project.

## **The Experience Learned from this Project**

From this project, we garnered lots of experience applicable to the real world. We translated business rules and assumptions into logical and relational views from external views. This process began by creating the business rules and assumptions that best aligned with the organization's goals. From here, we created an ER model that, to the best of our ability, represents the operations pertaining to the organization. Some examples include modelling relationships between users, properties, and transactions. Likewise, from the business rules, we derived DBMS operations that address real organizational needs through practical queries. For example, calculating the number of listings per owner to help identify "hot sellers" or analyzing message volumes for each listing to better gauge buyer interest. Moreover, through the creation of our database design, we gained a better understanding of the many JOIN operations. This includes simple INNER JOINS to more complex OUTER JOINS. Another experience we learned was the maintenance of data integrity at multiple levels. We enforced data integrity through careful assignment of primary keys (e.g. `account_id` for the account table, `address_id` for the address table, etc.). Referential integrity was enforced through various ON DELETE clauses (e.g. `CONSTRAINT fk_credit_card_account_id FOREIGN KEY(account_id) REFERENCES account(account_id) ON DELETE CASCADE`). Furthermore, domain

integrity was enforced through diligent type casting for data types (e.g. VARCHAR() for account first and last name, NUMERIC for credit card number, INT for message\_id, etc.). Through all these practical learning experiences, we created a reliable foundation for the organization's platform data operations.