Introduction

Canadian society faces serious problems with rising housing costs as regular citizens cannot afford suitable housing. For property management organizations the implementation of efficient solutions holds great importance because they allow balanced access to rental properties while simplifying lease transactions and managing governmental subsidies.

The Housing Management & Rental System stands as a solution to address property management problems through its complete database system. Continuous operations happen through this system to track tenant applications and lease data as well as property maintenance records and government subsidy oversight. The system brings together different stakeholders including landlords and tenants with property managers as well as government bodies to maintain efficient data flow and property control.

At least ten interconnected database tables are used for normalization which maintains proper relationships and achieves efficient query execution. The system includes built-in features that permit collectors to retrieve useful information for machine learning pipelines that evaluate rental market trends and handle payment patterns and state funding distribution.

The implementation shows how one uses SQL abilities to build schemas and execute DQL queries alongside DML instructions and administer databases. The system demonstrates scalable design capabilities as a result which positions it as an optimal solution for rental operation streamlining among housing organizations and property management firms.

Objective

This system seeks to build a complete database platform which welcomes rental property operation control together with tenant application handling and lease administration and government subsidy supervision. Through this system Canada targets multiple housing challenges by managing data efficiently and enabling better access to available housing properties along with optimized rental management. The key objectives are:

- To offer a structured format which enables landlords together with property managers to document and monitor property details as well as rental fees alongside availability data.
- Property inspections as well as maintenance problem tracking functions to operate without delays allowing for quality housing standards.
- Create an efficient relational structure for property, tenant, lease, and maintenance data management.
- Populate tables with realistic Canadian demographic, rental, and subsidy information for accurate analysis.
- Develop SQL queries and views to extract datasets for payment prediction, price optimization, and maintenance forecasting.
- Build a flexible system for future enhancements like automated procedures, indexing, and real-time reporting.

Entities and Their Functions

1. Landlord
    - Stores details of landlords who own rental properties.
    - Attributes: landlord_id (PK), first_name, last_name, phone, email.
    - Function: Manages property ownership and landlord contact details.
2. Property
    - Represents rental properties available in the system.
    - Attributes: property_id (PK), landlord_id (FK), address_id (FK), rental_price, is_affordable_housing.
    - Function: Tracks rental units, ownership, and pricing.
3. Location
    - Stores address information for properties.
    - Attributes: address_id (PK), street, city, province, postal_code.
    - Function: Normalizes property addresses for efficient location management.
4. Tenant
    - Stores details of tenants renting properties.
    - Attributes: tenant_id (PK), first_name, last_name, phone, email.
    - Function: Tracks tenant information and links to lease agreements.
5. Rental Application
    - Stores applications submitted by tenants for rental properties.
    - Attributes: application_id (PK), tenant_id (FK), property_id (FK), status, application_date.
    - Function: Tracks rental applications, their status (Pending, Approved, Rejected), and submission date.
6. Lease Agreement
    - Manages active rental contracts between tenants and landlords.
    - Attributes: lease_id (PK), tenant_id (FK), property_id (FK), start_date, end_date, monthly_rent, status.
    - Function: Tracks lease duration, rent, and active/expired contracts.
7. Payment

- Stores rental payments made by tenants.
- Attributes: payment_id (PK), lease_id (FK), amount_paid, payment_date.
- Function: Tracks rental payments and associates them with lease agreements.

8. Government Subsidy

    - Tracks government-funded rental assistance for eligible tenants.
    - Attributes: subsidy_id (PK), tenant_id (FK), amount, status, approval_date.
    - Function: Monitors financial assistance provided to tenants for affordable housing.

9. Inspection

    - Tracks property inspections conducted by authorities or landlords.
    - Attributes: inspection_id (PK), property_id (FK), inspector_name, inspection_date, notes.
    - Function: Maintains property inspection history and reports.

10. Maintenance Request

    - Stores tenant-submitted maintenance or repair requests.
    - Attributes: request_id (PK), tenant_id (FK), property_id (FK), request_date, description, status.
    - Function: Allows tenants to request repairs, track maintenance issues, and update request status.

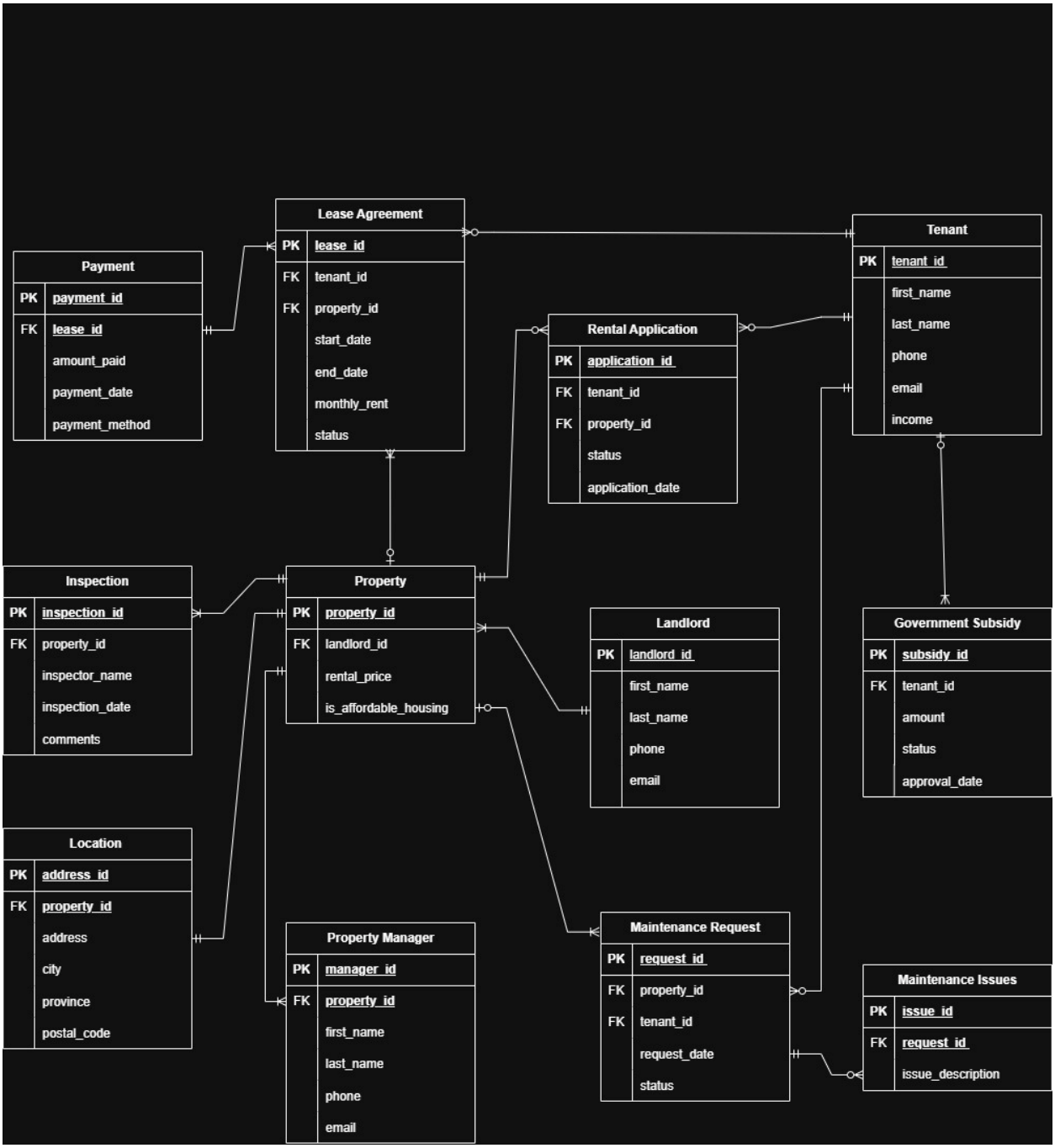
11. Maintenance Issue

    - Manages active maintenance work orders.
    - Attributes: issue_id (PK), request_id (FK), manager_id (FK), issue_description, assigned_date, resolved_date, status.
    - Function: Tracks maintenance assignments and work progress.

12. Property Manager

    - Stores details of property managers handling property operations.
    - Attributes: manager_id (PK), first_name, last_name, phone, email.
    - Function: Manages property-related issues, maintenance requests, and inspections

ER Diagram



File: https://drive.google.com/file/d/1YYXpDtJpvKvFu9EjSoXiPJEYJb3WpbBu/view?usp=sharing

The relationships between these tables were mapped by preparing an Entity-Relationship (ER) Diagram. That way, this diagram makes it easy to understand how data moves around the system as it connects orders to products and payments to customers. All the diagrams were done using Draw.io

Entity 1	Entity 2	Relationship Type	Description
Landlord	Property	One-to-Many	One landlord can own multiple properties, but each property is owned by only one landlord.
Tenant	Rental Application	One-to-Many	A tenant can apply for multiple properties, but each rental application is tied to one tenant.
Rental Application	Property	Many -to-One	Multiple applications can be submitted for a property, but each application is for a single property.
Tenant	Lease Agreement	One-to-Many	A tenant can have multiple lease agreements over time (e.g., renewals, different properties).
Lease Agreement	Property	Many -to-One	A lease is for one property, but a property can have multiple leases over time.
Lease Agreement	Payment	One-to-Many	A lease agreement can have multiple payments, but each payment belongs to one lease.
Tenant	Government Subsidy	One-to-Many	A tenant can receive multiple subsidies over time, but each subsidy is linked to only one tenant.
Tenant	Maintenance Request	One-to-Many	A tenant can submit multiple maintenance requests, but each request is tied to one tenant.
Property	Maintenance Request	One-to-Many	A property can have multiple maintenance requests submitted over time.
Maintenance Request	Maintenance Issues	One-to-Many	A maintenance request can be associated with multiple maintenance issues.
Property Manager	Property	One-to-Many	One property manager can oversee multiple properties, but each property is managed by only one property manager.
Property	Location	One-to-One	Each property has a unique location entry.
Property	Inspection	One-to-Many	A property can be inspected multiple times, but each inspection is for one property.

Data Definition Language (DDL)

The DDL (Data Definition Language) defines the structure of the Rental Management System database by creating schema, tables, relationships, views, and functions. It ensures data integrity using primary keys, foreign keys, and constraints.

- Tables such as Tenant, Lease Agreement, Property, Payment, Rental Application, Inspection, Maintenance Request, and Government Subsidy are created to store rental-related data.
- Views are created for active lease agreements, pending rental applications, government-subsidized tenants, and property inspections to simplify data access.
- User-Defined Functions (UDFs) calculate total rent paid, lease status, and subsidy eligibility, ensuring reusable logic and optimized queries.

Following are screenshots of tables, views, and functions:

```
/* Create the database */
DROP SCHEMA IF EXISTS housingrentalsystem;
CREATE SCHEMA housingrentalsystem;

-- Create Tables
CREATE TABLE landlord (
    landlord_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    phone VARCHAR(15) NOT NULL,
    email VARCHAR(100) NOT NULL
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE tenant (
    tenant_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    phone VARCHAR(15) NOT NULL,
    email VARCHAR(100) NOT NULL,
    income DECIMAL(10,2) NOT NULL
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```sql
CREATE VIEW LeasePaymentSummary AS
WITH PaymentSummary AS (
    SELECT
        lease_id,
        COUNT(payment_id) AS payments_made,
        COALESCE(SUM(amount_paid), 0) AS total_paid,
        MAX(payment_date) AS last_payment_date
    FROM Payment
    GROUP BY lease_id
)
SELECT
    l.lease_id,
    l.property_id,
    l.tenant_id,
    l.start_date,
    CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
    l.monthly_rent,
    COALESCE(ps.payments_made, 0) AS payments_made,
    COALESCE(ps.total_paid, 0) AS total_paid,
    ps.last_payment_date,
    COALESCE(
```

```sql
-------------------- Functions -----------------------------------------
DELIMITER //
CREATE FUNCTION CalculateTotalRentPaid(tenant_id INT)
RETURNS DECIMAL(10,2)
NOT DETERMINISTIC
DETERMINISTIC SQL SECURITY INVOKER
BEGIN
    DECLARE total DECIMAL(10,2);
    SELECT IFNULL(SUM(p.amount_paid),0) INTO total
    FROM Payment p
    JOIN Lease_Agreement la ON p.lease_id = la.lease_id
    WHERE la.tenant_id = tenant_id;
    RETURN total;
END;
// DELIMITER ;
```

Data Manipulation Language (DML)

The DML (Data Manipulation Language) is used in this project to manage data by performing operations like INSERT, UPDATE, and DELETE on tables. These operations allow adding new records, modifying existing data, and removing outdated information. DML ensures that the system's data remains accurate and up to date, supporting dynamic interactions with rental-related information. Below are some snippets of DML Quries used in our system:

```
-- Insert Data in Landlord table
INSERT INTO landlord (first_name, last_name, phone, email) VALUES
('James', 'Smith', '416-555-0101', 'james.smith@mail.com'),
('Sophia', 'Johnson', '604-555-0202', 'sophia.j@mail.ca'),
('Liam', 'Tremblay', '514-555-0303', 'liam.t@qc.ca'),
('Emma', 'Brown', '403-555-0404', 'emma.b@ab.com'),
('Noah', 'Roy', '780-555-0505', 'noah.roy@edm.ca'),
('Olivia', 'Gagné', '418-555-0606', 'olivia.g@quebec.qu'),
('Lucas', 'Wilson', '902-555-0707', 'lucas.w@ns.ca'),
('Charlotte', 'MacDonald', '506-555-0808', 'charlotte.m@nb.ca'),
('Benjamin', 'Li', '905-555-0909', 'ben.li@ontario.ca'),
('Amelia', 'Singh', '519-555-1010', 'amelia.s@kw.ca'),
('Mason', 'Taylor', '250-555-1111', 'mason.t@bc.ca'),
('Ava', 'Campbell', '306-555-1212', 'ava.c@sk.sk.ca'),
('Elijah', 'White', '204-555-1313', 'elijah.w@mb.ca'),
('Harper', 'Lee', '867-555-1414', 'harper.l@yk.ca'),
('Evelyn', 'Martin', '709-555-1515', 'evelyn.m@nl.ca'),
('Logan', 'Thompson', '867-555-1616', 'logan.t@nt.ca'),
('Ella', 'Scott', '902-555-1717', 'ella.s@pei.ca'),
('Oliver', 'Bouchard', '581-555-1818', 'oliver.b@qc2.ca'),
('Mia', 'Patel', '705-555-1919', 'mia.p@north.on.ca'),
('Aiden', 'Ouellet', '819-555-2020', 'aiden.o@outaouais.qc.ca');
```

```
-- Update statement for temporary adjustment by adding approved subsidy amounts
UPDATE tenant t
JOIN Government_Subsidy gs ON t.tenant_id = gs.tenant_id
SET t.income = t.income + gs.amount
WHERE gs.status = 'Approved';

-- Delete statement for leases that are terminiated
DELETE FROM lease_agreement
WHERE status = 'Terminated' and lease_id = 8;
```

Data Query Language (DQL)

In this project, DQL (Data Query Language) is used to extract some reports and datasets for Machine Learning analysis. We performed analyses such as tenant payment behavior prediction (e.g., delinquency risk), rental price optimization (e.g., suggested price adjustments), and maintenance cost forecasting (e.g., predicted monthly expenses). These analyses provide actionable insights, such as identifying high-risk tenants, recommending optimal rental prices, and predicting maintenance budgets, enabling data-driven decision-making for property management. Below showcasing the queries used for these analyses.

1. Tenant payment Report

```sql
-- Finding report of tenants who paid three month rent
SELECT * FROM LeasePaymentSummary
WHERE total_paid >= monthly_rent * 3;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

lease_id	property_id	tenant_id	start_date	tenant_name	monthly_rent	payments_made	total_paid	last_payment_date	months_covered
1	1	1	2023-01-01	Léa Gagnon	2000.00	3	6000.00	2023-03-05	3
15	10	15	2023-05-15	Zoe Wilson	3600.00	3	10800.00	2023-07-20	3

2. Finding Underperforming properties

```sql
-- Fetching result for underperforming properties
SELECT * FROM (
    SELECT
        p.property_id,
        loc.address,
        loc.city,
        loc.province,
        loc.postal_code,
        p.rental_price,
        COUNT(DISTINCT CASE WHEN l.status = 'Active' THEN l.lease_id END) AS active_leases,
        COALESCE(SUM(py.amount_paid), 0) AS total_income,
        COUNT(mr.request_id) AS maintenance_count,
        ROUND(COALESCE(SUM(py.amount_paid), 0) / NULLIF(p.rental_price, 0), 1) AS occupancy_rate
    FROM Property p
    JOIN Location loc ON p.property_id = loc.property_id
    LEFT JOIN Lease_Agreement l ON p.property_id = l.property_id
    LEFT JOIN Payment py ON l.lease_id = py.lease_id
    LEFT JOIN Maintenance_Request mr ON p.property_id = mr.property_id
    GROUP BY p.property_id, loc.address, loc.city, loc.province, loc.postal_code, p.rental_price
    ) A
WHERE occupancy_rate < 5 AND maintenance_count > 2;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

property_id	address	city	province	postal_code	rental_price	active_leases	total_income	maintenance_count	occupancy_rate
1	100 Queen St W	Toronto	ON	M5H 2N2	2205.00	1	6000.00	3	2.7
10	1000 Bank St	Ottawa	ON	K1S 5K8	3600.00	1	10800.00	3	3.0

3. Property Price Rental Analysis

```sql
SELECT
    p.property_id,
    loc.address,
    loc.city,
    loc.province,
    loc.postal_code,
    p.rental_price,
    (SELECT AVG(p2.rental_price)
     FROM Property p2
     JOIN Location loc2 ON p2.property_id = loc2.property_id
     WHERE loc2.city = loc.city) AS avg_city_price,
    p.rental_price - (SELECT AVG(p3.rental_price)
                      FROM Property p3
                      JOIN Location loc3 ON p3.property_id = loc3.property_id
                      WHERE loc3.city = loc.city) AS price_deviation,
    RANK() OVER (PARTITION BY loc.city
                 ORDER BY p.rental_price DESC) AS city_price_rank,
    COUNT(mr.request_id) OVER (PARTITION BY p.property_id) AS maintenance_count
FROM Property p
JOIN Location loc ON p.property_id = loc.property_id
LEFT JOIN Maintenance_Request mr ON p.property_id = mr.property_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

property_id	address	city	province	postal_code	rental_price	avg_city_price	price_deviation	city_price_rank	maintenance_count
1	100 Queen St W	Toronto	ON	M5H 2N2	2205.00	1977.500000	227.500000	1	1
2	200 Granville St	Vancouver	BC	V6C 1S6	3200.00	3266.666667	-66.666667	2	1
3	300 Rue Sainte-Catherine	Montréal	QC	H3B 4W5	1850.00	2025.000000	-175.000000	2	1
4	400 8th Ave SW	Calgary	AB	T2P 1E5	2750.00	2800.000000	-50.000000	2	1
5	500 Barrington St	Halifax	NS	B3J 1Z1	2047.50	2023.750000	23.750000	1	1

Result 45

4. Tenant Payment Behavior Prediction

```sql
-- Tenant Payment Behavior Prediction
WITH PaymentFeatures AS (
    SELECT
        t.tenant_id,
        t.income,
        AVG(DATEDIFF(p.payment_date, la.start_date)) AS avg_payment_delay,
        COUNT(DISTINCT p.payment_id) AS total_payments,
        COUNT(mr.request_id) AS maintenance_requests,
        DATEDIFF(CURDATE(), MAX(p.payment_date)) AS days_since_last_payment,
        CASE WHEN DATEDIFF(CURDATE(), MAX(p.payment_date)) > 30 THEN 1 ELSE 0 END AS is_delinquent
    FROM Tenant t
    JOIN Lease_Agreement la ON t.tenant_id = la.tenant_id
    LEFT JOIN Payment p ON la.lease_id = p.lease_id
    LEFT JOIN Maintenance_Request mr ON t.tenant_id = mr.tenant_id
    GROUP BY t.tenant_id
)
SELECT
    *,
    ROUND(income / la.monthly_rent, 2) AS rent_to_income_ratio
FROM PaymentFeatures
JOIN Lease_Agreement la USING (tenant_id);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

tenant_id	income	avg_payment_delay	total_payments	maintenance_requests	days_since_last_payment	is_delinquent	lease_id	property_id	start_date	end_date	monthly_rent	status
1	42500.00	34.0000	3	3	750	1	1	1	2023-01-01	2024-01-01	2000.00	Active
2	68750.00	5.0000	1	1	735	1	2	3	2023-03-15	2024-03-15	1800.00	Active
3	53000.00	4.0000	1	1	778	1	3	5	2023-02-01	2024-02-01	1900.00	Termin
4	49900.00	4.0000	1	1	719	1	4	7	2023-04-01	2024-04-01	1750.00	Active

5. Differentiating tenant payment consistency using UDFs

```sql
-- Compare tenant payment behavior using UDFs
SELECT
    tenant_id,
    CalculateTotalRentPaid(tenant_id) AS total_paid, -- user defined function
    IsConsistentPayer(tenant_id) AS consistency -- user defined function
FROM Tenant
WHERE income > 50000;
```

Result Grid | Filter Rows:

tenant_id	total_paid	consistency
2	1800.00	Yes
3	1900.00	Yes
5	2050.00	Yes
6	2200.00	Yes
7	3600.00	No

Conclusion

This project focused on designing and implementing a property management database system to streamline rental operations, tenant tracking, and maintenance management. By leveraging SQL DDL, DML, and DQL, we created a robust schema, populated it with realistic Canadian data, and performed advanced analytics for Machine Learning (ML) use cases such as tenant payment behavior prediction, rental price optimization, and maintenance cost forecasting. Future enhancements include creating stored procedures for automated data processing, optimizing queries with indexing, and integrating real-time dashboards for better decision-making. Additionally, implementing data validation rules and backup procedures will further enhance system reliability and performance.