INSY 661 - Databases & Distributed Systems

RENTALS.CA

FINAL PROJECT

By Audrey Delisle, Xingchen Luo, Sheida Majidi,
Meriem Mehri, Xinran Yu
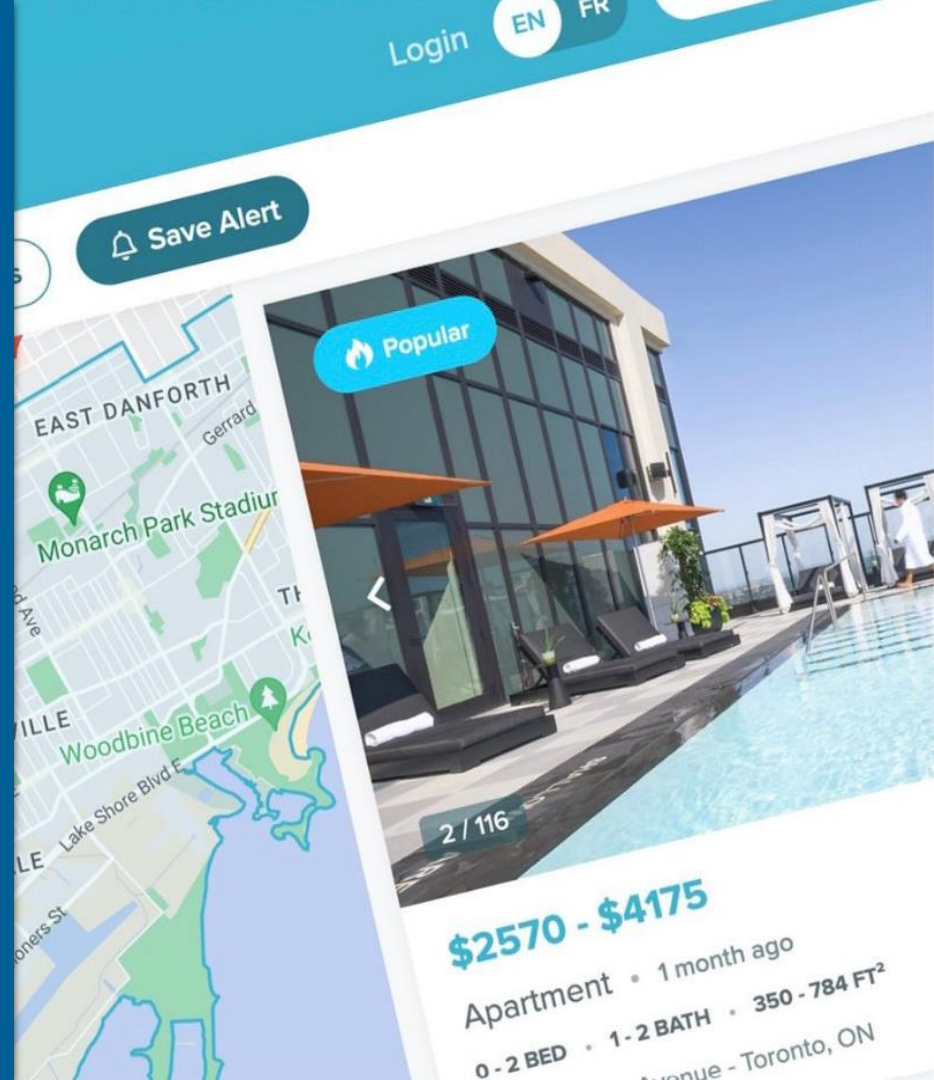
Rentals.ca 2

BUSINESS SCENARIO
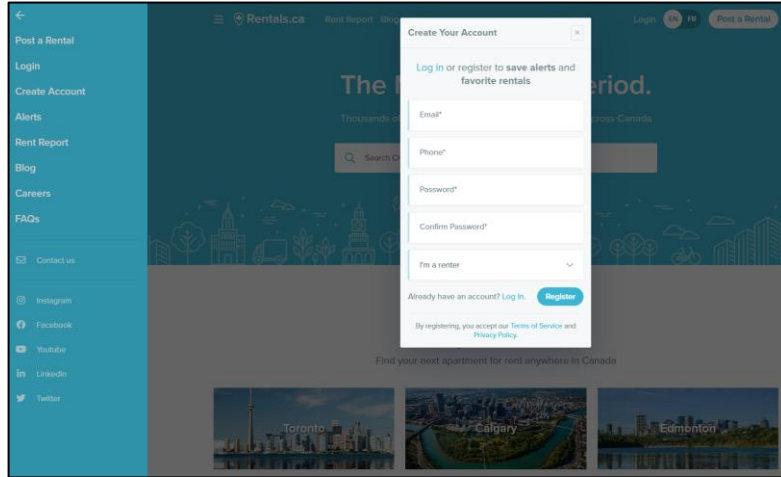
01

BUSINESS PROCESSES

Rentals.ca is a website that provides rental accommodation listings across Canada. The business aims to help tenants find their ideal place to live, and landlords to advertise their properties to potential renters.

The business processes include:
- **Posting a rental:** Landlords can create an account and post their rental properties on the website, with details such as location, price, size, amenities, and photos.
- **Searching for a rental:** Tenants can browse through thousands of apartments, houses, and condos for rent across Canada, using filters such as city, neighborhood, address, or ad number.
- **Rent report:** Rentals.ca publishes a monthly rent report that analyzes the average asking rents in Canada and various provinces and cities, based on data from purpose-built and condominium apartments.
- **Blog:** Rentals.ca also maintains a blog that features articles on topics such as renting tips, market news, lifestyle, and design.

FUNCTIONALITIES OF WEBSITE

User Profile Creation

Unit Features & Amenities

FUNCTIONALITIES OF WEBSITE
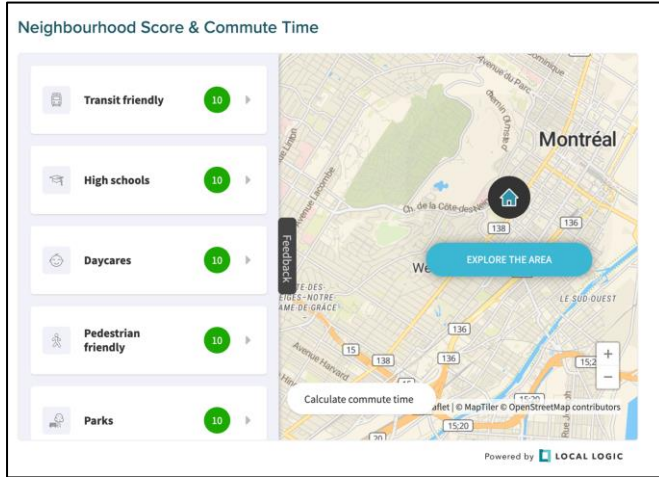
Building Rating Score
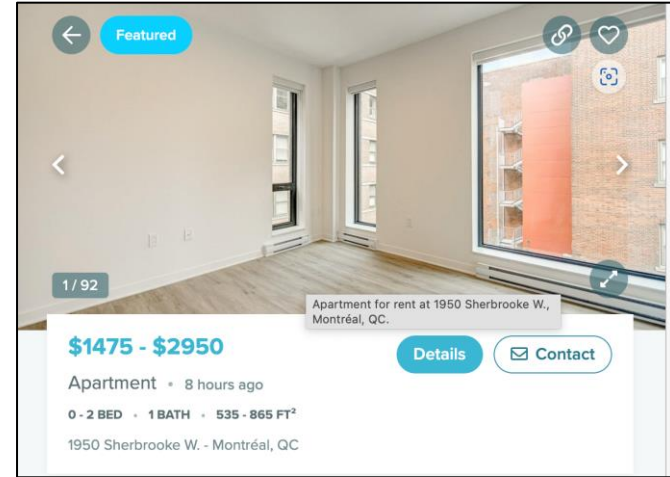
Building/Unit Info

Rentals.ca 6

MISSION STATEMENT & OBJECTIVE

MISSION STATEMENT

The Rentals.ca database system serves as the backbone of a dynamic, secure, and scalable national apartment rental platform. **Its key mission is to provide instant listing updates, precise property details, and tailored user suggestions.** Emphasizing data integrity and privacy, its integration enhances user engagement, refines search capabilities, and streamlines the rental process. By delivering a high-performing and reliable solution, Rentals.ca connects individuals with their ideal living spaces, fostering trust and convenience among tenants, property owners, and lenders across the country.

OBJECTIVES

| Database Objective	Required Resources/Activities
To support instantaneous updates and availability of apartment listings.	*Synchronization between listing agents, property owners and the website to ensure an efficient listing management.*
To store and maintain precise information regarding the properties' details, features, amenities, location and lease terms.	*Adequate storage and data validation mechanisms to preserve accuracy.*
To develop workflows simplifying the apartment renting process, from listing selection to lease agreement.	*Adequate budget will be allocated for process optimization, user testing, and continuous improvement efforts.*
To ensure uninterrupted service availability thanks to a high-performance server infrastructure.	*Adequate financial resources will be dedicated to hardware upgrades and regular server performance testing.*
To safeguard sensitive information and establish encryption mechanisms & secured authentication processes.	*Adequate investment will be allocated to invest in robust cybersecurity tools.*

ENTITY RELATIONSHIP DIAGRAM (ERD)

03

ERD
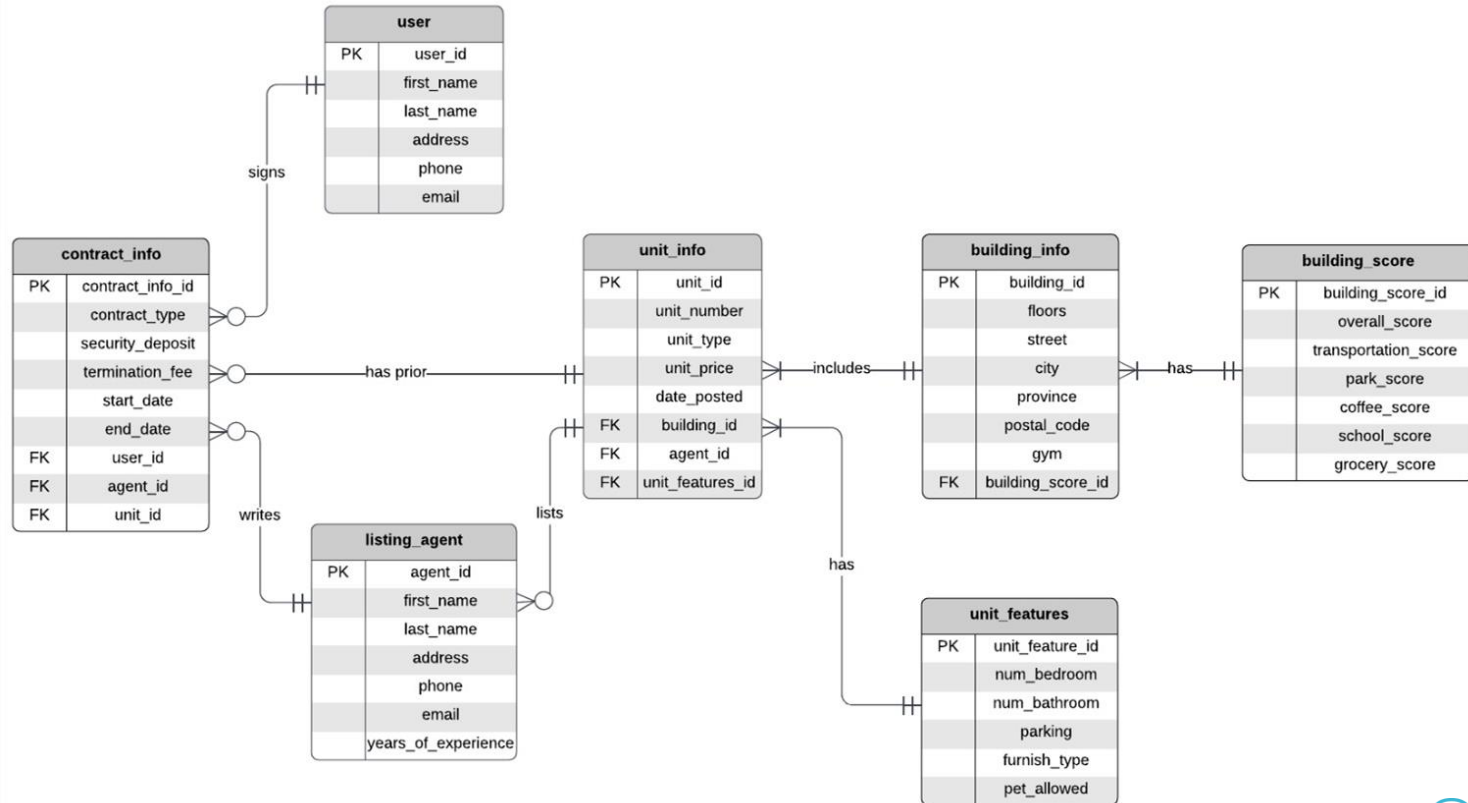
RELATIONAL SCHEMA

04

RELATIONAL SCHEMA

User (user_id, first_name, last_name, address, phone, email)
Primary Key: user_id

Unit_Info (unit_id, unit_number, unit_type, unit_price, area, date_posted, agent_id, unit_features_id, building_id)
Primary Key: unit_id
Foreign Key: agent_id References Listing_Agent (agent_id)
Foreign Key: unit_features_id References Unit_Features (unit_features_id)
Foreign Key: building_id References Building_Info (building_id)

Unit_Features (unit_feature_id, num_bedroom, num_bathroom, furnish_type, pet_allowed, parking)
Primary Key: unit_feature_id

Building_Info (building_id, floors, street, city, province, postal_code, gym, building_score_id)
Primary Key: building_id
Foreign Key: building_score_id References Building_Score (building_score_id)

RELATIONAL SCHEMA

Building_Score (building_score_id, overall_score, transportation_score, park_score, coffee_score, school_score, grocery_score)
Primary Key: bulding_score_id

Listing_Agent (agent_id, first_name, last_name, address, phone, email, years_of_experience)
Primary Key: agent_id

Contract_Info (contract_info_id, unit_id, contract_type, security_deposit, termination_fee, start_date, end_date, user_id, agent_id)
Primary Key: contract_info_id
Foreign Key: unit_id References Unit_Info (unit_id)
Foreign Key: user_id References User (user_id)
Foreign Key: agent_id References Listing_Agent (agent_id)

QUERIES - PURPOSE & EXPLANATION

OBJECTIVES

Database Objective	Required Resources/Activities
To maintain a user-friendly interface for intuitive navigation, and efficient search functionalities.	*Appropriate frontend and backend development resources will be allocated for continuous UI/UX enhancements on Rentals.ca website.*
To offer customized recommendations tailored to the users' needs and preferences in terms of budget, location, history, interactions, etc.	*Development of an algorithmic framework using machine learning tools for personalized recommendations based on user preferences and interactions.*
To foster user trust and convenience by facilitating transparent communication between tenants, property owners, and lenders.	*Sufficient budget for customer service enhancements and ongoing support resources.*
To ensure scalability of the database to accommodate a growing volume of listings and users across the country.	*Adequate financing for server scaling, database optimization, and network expansion.*

QUERY 1: OBJECTIVE

Objective: The team at rentals.com wants to provide **personalized suggestions** to their returning clients. To achieve this, the website curates a list of units available in buildings that have similar ratings (or scores) to those the user "UID5" has previously contracted. This way, the user gets **recommendations based on their historical preferences**, increasing the chances of them making another rental or purchase decision. The query ensures that previously contracted units by the user are not shown again, providing a fresh set of recommendations every time.

QUERY 1: EXPLANATION

Main Query:
We start by selecting the desired fields from the Unit_Info table.
We join with the Building_Info table on building_id to get building details.
Our main filtering criterion is to ensure that the building's score id is within the scores of buildings that the user "UID5" had contracts in.
We also ensure that the unit hasn't been previously contracted by the user "UID5".
Lastly, we order the results by the date_posted in descending order, giving priority to the most recently posted units.

Subquery 1 (building_score_id):
This subquery retrieves the building score IDs of buildings where the user "UID5" previously made contracts.
To achieve this, we join Contract_Info, Unit_Info, and Building_Info tables.
We filter out the results where the user_id is "UID5".
We get distinct building_score_id since the user can have multiple contracts in the same building or in buildings with the same score.

Subquery 2 (units previously contracted):
This subquery fetches all unit IDs that the user "UID5" previously made contracts for.
We then select from the Contract_Info table filtering by user_id as 'UID5'.

QUERY 1: CODE & OUTPUT

Code:

```
SELECT ui.unit_id, ui.unit_number, ui.unit_price,
ui.date_posted, ui.unit_type
FROM Unit_Info ui
INNER JOIN Building_Info bi ON ui.building_id = bi.building_id
WHERE bi.building_score_id IN (
    SELECT DISTINCT b.building_score_id
    FROM Contract_Info ci
    INNER JOIN Unit_Info ui ON ci.unit_id = ui.unit_id
    INNER JOIN Building_Info b ON ui.building_id =
b.building_id
    WHERE ci.user_id = 'UID5'
)
AND ui.unit_id NOT IN (
    SELECT unit_id
    FROM Contract_Info
    WHERE user_id = 'UID5'
)
ORDER BY ui.date_posted DESC;
```

Showing rows 0 - 2 (3 total, Query took 0.0099 seconds.) [date_posted: 2023

`SELECT ui.unit_id, ui.unit_number, ui.unit_price, ui.date_posted,`
`ui ON ci.unit_id = ui.unit_id INNER JOIN Building_Info b ON ui.bu`

unit_id	unit_number	unit_price	date_posted ▼ 1	unit_type
UIID7	G105	1250.00	2023-07-30	Apartment
UIID14	N205	1750.00	2023-07-25	Apartment
UIID15	O302	2350.00	2023-06-10	Condo

Back   Print

QUERY 2: OBJECTIVE

Objective: Rentals.com wishes to analyze the "time-to-rent" for various properties. They want to understand the **duration for which units remain vacant between rental contracts**. This can offer insights into:

- **Market Demand:** Shorter time-to-rent indicates high demand in the region or for the unit type.

- **Pricing Dynamics:** Longer days without tenancy might hint at units being overpriced or having other issues.

- **Geographical Preferences:** By breaking down the data by province and city, Rentals.com can analyze which areas have properties that get rented faster than others.

QUERY 2: EXPLANATION

Main Query:

Begins with the selection of pertinent columns from the Unit_Info table (referred to as u): This includes the unit's ID (unit_id) and the date it was listed (date_posted). The resulting data set is ordered in descending order based on the days_without_tenancy column and the query's results are limited to the top 10 records.

Joins:

A join operation with Contract_Info (abbreviated as c) enables the extraction of the start_date of the lease, giving an idea about when the unit began its tenancy.
Subsequent joining with Building_Info (aliased as bi) provides geographic details related to the unit, notably the province and city.

Calculating Time-to-Rent:

The GREATEST function paired with DATEDIFF is utilized to determine the time difference (in days) between when the unit was listed and when its contract began. Should a lease commence before its listing date, this is considered a data inconsistency. Here, the GREATEST function ensures any negative durations are reset to zero, symbolizing zero days without tenancy.

QUERY 2: CODE & OUTPUT

Code:

```
SELECT
    u.unit_id,
    u.date_posted AS unit_posted_date,
    c.start_date AS contract_start_date,
    GREATEST(DATEDIFF(c.start_date, u.date_posted), 0) AS
days_without_tenancy,
    bi.province,
    bi.city
FROM
    Unit_Info u
JOIN
    Contract_Info c ON u.unit_id = c.unit_id
JOIN
    Building_Info bi ON u.building_id = bi.building_id
ORDER BY
    days_without_tenancy DESC
LIMIT 10;
```

✔ Showing rows 0 - 9 (10 total, Query took 0.0123 seconds.) [days_without_tenancy: 87... - 31...]

```
SELECT u.unit_id, u.date_posted AS unit_posted_date, c.start_date AS contract_start_date, GREAT
ON u.building_id = bi.building_id ORDER BY days_without_tenancy DESC LIMIT 10;
```

unit_id	unit_posted_date	contract_start_date	days_without_tenancy ▽ 1	province	city
UIID3	2023-06-25	2023-09-20	87	QC	Montreal
UIID9	2023-06-15	2023-08-30	76	QC	Quebec City
UIID6	2023-06-20	2023-08-25	66	ON	Ottawa
UIID6	2023-06-20	2023-08-15	56	ON	Ottawa
UIID9	2023-06-15	2023-08-05	51	QC	Quebec City
UIID4	2023-07-05	2023-08-20	46	AB	Calgary
UIID8	2023-08-05	2023-09-10	36	MB	Winnipeg
UIID5	2023-08-10	2023-09-10	31	AB	Edmonton
UIID1	2023-08-01	2023-09-01	31	ON	Toronto
UIID8	2023-08-05	2023-09-05	31	MB	Winnipeg

LEARNING EXPERIENCE

90

ISSUES & CHALLENGES

1) **No Overlapping Data:**
- Issue: Data between tables didn't have overlapping entries, which meant JOIN operations returned empty results.
- Solution: Introduced a data verification and seeding process to ensure consistent and overlapping data entries across tables, thereby ensuring that JOIN operations produced meaningful results.

1) **DBFiddle Troubles:**
- Issue: Encountered unexpected behaviors and issues while using DBFiddle for testing and demonstrating SQL queries.
- Solution: Switched to a more robust local SQL environment (MAMPS) for testing and then validated the results on DBFiddle. This also enabled better version control and troubleshooting capabilities.

1) **Unique ID Confusion:**
- Issue: Using the same unique IDs (like 1,1,1) across multiple tables made it confusing during JOIN operations and data analysis.
- Solution: Implemented a clear unique ID naming and assignment convention. For instance, prefixing IDs based on their table (e.g., BID1 for Building ID in Building_Info, UID1 for User ID in User_Info) to distinguish them easily.

LESSONS LEARNT

- **Consistent Data Seeding:** Ensuring consistency and meaningful relations between datasets is crucial when setting up a relational database.

- **Tool Reliability:** While online tools like DBFiddle are convenient, relying solely on them can be limiting. It's essential to have a backup plan or a local environment to validate results.

- **Naming Conventions:** Proper naming conventions, especially in databases with multiple relational tables, can significantly reduce confusion and streamline the data analysis process.

IDEAS TO EXTEND

1) **Data Validation Tool:** Develop a tool that periodically checks data integrity and consistency across tables, alerting administrators to any discrepancies.
2) **Enhanced Query Interface:** Design a user-friendly interface where non-technical stakeholders can generate custom reports without writing SQL queries.
3) **Database Optimization:** Regularly review and optimize the database schema, indexing frequently queried columns and archiving old data to improve performance.
4) **Integrate Machine Learning:** Based on the rich data available, incorporate machine learning models to predict rental trends, tenant preferences, and even potential maintenance needs for properties.