Report

APAN 5310 Project Report

Team Members: Yuxuan Chen, Yishan Liu, Xianghui Meng, Haoxiang (Steven) Huo

Overview

Dream Homes NYC is a real estate company that assists people in buying and selling luxury homes in New York City. Our team of database specialists has been invited to address two primary challenges. First, we will transform years of archived, unstructured data—ranging from legacy spreadsheets to scanned documents—into a clean, well-organized repository that supports efficient search and analysis. Second, we will implement an automated pipeline to combine and merge multiple real-time data sources into this unified system, eliminating the need for any manual data transfers. As all of our group members are international students who have experienced the challenges of finding and renting housing in New York City, we're excited to explore and find a solution to handle the large volume of data across properties, clients, and agents.

We chose to work on Dream Homes NYC because its centralized database is complex, and we're passionate about using technology to simplify real-world challenges. We're motivated by the opportunity to apply our database expertise to turn chaotic information into clear, actionable insights, helping organizations make faster, smarter decisions and reduce the workload of data tasks for employees. To ensure we deliver the best solution to the goal, we began by sharing a small, cleaned sample of our combined property, client, and transaction data with a link to the full dataset on GitHub, and explaining why it reflects Dream Homes NYC's real-world needs. Then we'll explain step by step through our database design, complete with a PDF ER diagram that includes all of the relationships and cardinality between tables, which were created through SQL code. Next, we'll detail our automated ETL process in Python that imports data from old records and current data sources, explains every transformation choice, and loads everything into our new schema. Once the data is in place, we'll show advanced analyses with the exact SQL or Python code and the resulting insights to answer those key business questions. We'll also explain how analysts can query the database directly (or via Python/R) while managers and executives can view clear, automatically refreshed dashboards in Metabase. Finally, we'll cover our plans for database performance, redundancy, and a cloud-based deployment, then wrap up with a concise conclusion that highlights how our relational database, ETL pipeline, and analytics work together to give Dream Homes NYC faster, smarter decision-making and a single source of truth.

This unified database will reduce manual errors, eliminate duplicate records, and save time spent on data cleanup. Brokers will be able to price listings based on the latest comparable sales

in minutes, rather than days, and marketing teams can see in real-time which channels are driving inquiries, allowing them to reallocate budgets on the fly. Managers and executives will have clear, automatically updated dashboards that show revenue trends, agent performance, and neighborhood activity in a short time, letting them spot problems or opportunities as they emerge. Analysts will spend less time wrangling files and more time uncovering insights, like which client segments are most likely to convert or which amenities boost sale prices, while non-technical staff can rely on intuitive dashboards.

Task Division

Category	**Task**	**Assigned Member(s)**	**Planned DDL**
Business Understanding	Define detailed business requirements and client goals	Yuxuan Chen	Week 8
Database Schema	Design a normalized ERD (20+ tables in 3NF), including entities and relationships	Yishan Liu	Week 8
Data Simulation	Generate and clean sample data simulate messy or semi-structured input	Steven Huo	Week 8
Table Creation	Write SQL DDL scripts to define all tables and constraints in PostgreSQL	Xianghui Meng	Week 8
Data Loading	Create Python/SQL scripts to load data into the database and write code for analytical procedure	Xianghui Meng	Week 9
Complex Queries	Write 8 complex SQL queries to support business decisions	Yuxuan Chen	Week 10
Dashboards	Design and build interactive dashboards using Metabase or Power BI	Steven Huo	Week 11
Customer Interaction Plan	Write Customer Interaction Plan	Yishan Liu	Weak 11

Final Report	Co-author project report, including intro, schema design, queries, and business insights	All Members	Week 12
Final Presentation	Prepare slides and live demo walkthrough; each member presents part of the work	All Members	Week 12

Business Insights

Dream Homes NYC has asked us to build a system that makes their operations faster, keeps every record accurate, and delivers the best experience for both agents and clients. At its core, the database will track each office's profile—including its name, location, contact information, operating hours, and current manager—while maintaining a comprehensive employee directory that records employment status, departmental assignment, device allocation, and reporting relationships. It will also support detailed client records by capturing personal and contact data, recording individual budget ranges, structuring property preferences in a dedicated table, and preserving client feedback entries in a separate log. Every property listing will be modeled in full, with address details, listing characteristics, standardized amenities, current status, listing date, and all related media assets such as photographs and floor plans.

Beyond basic record-keeping, the system will document every scheduled appointment—identifying the agent, client, property, date, time, and outcome—and will record all transactions through distinct tables for sales, rentals, payments, and leases. Commissions will be computed and stored automatically, ensuring transparent earnings tracking for agents. Normalization of data into interrelated tables gives Dream Homes NYC real-time insight into office performance, agent productivity, transaction progress, and client objectives—empowering data-driven decisions and finely tuned, personalized property suggestions. Clients in turn enjoy a more responsive experience, receiving proactive, goal-aligned recommendations and timely updates that keep them informed and confident throughout their real estate journey.

Database Design (with ER-Diagram)

This schema follows a strict Third Normal Form design to eliminate redundancy and ensure integrity. Office and Employee tables capturing where the work happens and who's doing it, Client and ClientRole tracking every home-seeker and whether they're buyers, sellers, renters

or landlords, and PropertyType alongside Property defining each listing's category and its full address, price, and status. Detailed attributes live in dedicated tables: PropertyFeature for amenities, PropertyMedia for photos and floor plans; while Appointment logs every showing, tying together agents, clients, and properties with timestamps. Transactional workflows then span Transaction and Commission for sales and fee splits, Lease and PaymentRecord for rentals and payments, and MarketingCampaign with ClientLead to manage outreach and new inquiries. Finally, the Document table ensures every contract, inspection report, or appraisal is linked back to its property or deal, so no file ever slips through the cracks .

This structure not only prevents data duplication and update anomalies, but it also makes relationships clear: foreign keys enforce, for example, that every client's assigned_agent_id matches an existing employee, that each property belongs to a valid office and listing agent, and that every lease or payment ties back to real clients and staff. By giving each table a single responsibility and stitching them together with well-defined keys and constraints, Dream Homes NYC gains a flexible, scalable backbone for richer customer interactions—whether generating personalized follow-up reminders, aggregating a client's viewing history, or rolling out new features like maintenance requests without ever rewriting existing tables ans_psql.

Example Tables:

CLIENTS TABLE:

1. **client_id: INT** Numeric IDs fit in a 32-bit range. Declared **PRIMARY KEY** and **NOT NULL** to uniquely identify each client.

2. **full_name: VARCHAR(200) NOT NULL** 200 characters accommodate long personal or company‑contact names. **NOT NULL** because every client record needs a name.

3. **client_type: VARCHAR(20) NOT NULL** Values like "individual" or "company." 20 chars cover common labels. **NOT NULL** to enforce knowing who/what the client is.

4. **email: VARCHAR(100) NOT NULL UNIQUE** Email addresses seldom exceed 100 chars. **NOT NULL** because we require contact info, **UNIQUE** to prevent duplicates.

5. **phone: VARCHAR(20) NULL** Allows formatting characters, country codes, extensions. **NULL** because not every client may provide a phone number.

6. **assigned_employee_id: INT NOT NULL FOREIGN KEY → employees.employee_id**. Enforces that each client is managed by a valid employee.**NOT**

NULL because every client must have an assigned agent.

7. **created_at: TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP**
   Records when the client was added. **NOT NULL** so there's always a creation timestamp.

Relationship: Many **clients** → one **employee** (each client is handled by exactly one employee; an employee may serve many clients).

EMPLOYEES TABLE:

1. **employee_id: INT** Fits in a 32-bit integer. **PRIMARY KEY** and **NOT NULL** to uniquely reference each staff member.

2. **first_name: VARCHAR(50) NOT NULL**, **last_name: VARCHAR(50) NOT NULL**
   50 chars handle most personal names. **NOT NULL** because both parts of the name are required.

3. **email: VARCHAR(100) NOT NULL UNIQUE** Company emails rarely exceed 100 chars. **UNIQUE** to avoid duplicate accounts, **NOT NULL** for contact.

4. **phone: VARCHAR(20) NULL** Same reasoning as clients—optional contact.

5. **hire_date: DATE NOT NULL** Stores the date they joined; **NOT NULL** to track tenure.

6. **office_id: INT NOT NULL FOREIGN KEY → offices.office_id**.
   Ensures every employee belongs to an existing office. **NOT NULL** because every employee must be assigned to an office.

7. **manager_id: INT NULL FOREIGN KEY → employees.employee_id** (self-reference).
   NULL for top-level managers who report to no one.

8. **created_at: TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP**
   Timestamp of when the employee record was created; always present.

Relationship: One **employee** → many **clients** (via assigned_employee_id), and hierarchical reporting within **employees** (via manager_id).

ER Diagram:

https://lucid.app/lucidchart/7503e903-67a6-4ec5-a783-1eb32a0a3aed/edit?viewport_loc=-431%2C-247%2C9488%2C4103%2C0_0&invitationId=inv_cdf8abc9-9e77-4379-b06e-7f32298eb5fa

Data Simulation:

ETL Process

Our ETL is a single Python module, EnhancedDreamHomesETL, that reads the master CSV (dream_homes_nyc_dataset_v8.csv) and loads it into dream_homes_db on PostgreSQL. The module converts semi-structured records into typed rows that fit our schema and keeps natural keys stable so the job is safe to re-run. This section documents the schema context we load into, the cleaning done during extraction, the normalization decisions in the Transform stage, and the exact loading logic for each table group.

0) Schema context for loading

The schema is in third normal form and the ETL respects its boundaries. Offices manage employees (Office→Employee, 1:M). A client's master profile is stored once and roles are attached separately (Client→ClientRole, 1:M), because the same person can be a buyer in one deal and a landlord in another. Property categories are standardized in PropertyType; each Property is keyed by MLS number and owns many features and media items (Property→PropertyFeature/PropertyMedia, 1:M). Appointment records are time-stamped facts that connect an agent, a client, and a property. Commercial activity is unified in Transaction, with Commission as a 1:1 extension. Rentals add Lease (1:1 from Transaction) and many PaymentRecord rows (Lease→PaymentRecord, 1:M). Document and MarketingCampaign attach to properties or deals. These cardinalities determine our insert order and the natural keys we use for deduplication.

1) Data source and extract

We read the CSV into a pandas DataFrame with UTF-8 decoding; empty cells become NaN. Extraction is non-destructive: we keep the original tokens and rely on typed converters and mappers in Transform/Load to handle irregular text. We do not pre-drop duplicate lines; idempotent inserts on natural keys prevent duplication of base entities.

2) Cleaning rules (and how we treat unusual values)

Type coercion happens first and never crashes the batch. The helpers safe_decimal, safe_int, and safe_date convert values and return NULL on failure; each failure is logged as a WARNING. We rely on database constraints for sanity (for example, non-negative monetary amounts where enforced by DDL). When essential monetary fields for a deal are missing, the Transaction insert for that row is skipped, while the related Property still loads if its required fields parse.

3) Transform: normalization decisions and functions

The archived files contain composite strings and free-text categories. The Transform stage breaks them into atomic columns, maps labels to enums, and expands lists into child rows, so the database can enforce constraints and analytics do not rely on ad-hoc regex filters.

- Address parsing. parse_address splits a full address into address, city, state, zip_code using a right-to-left rule to fix state/ZIP and capitalization cues for city vs street. For Office inserts, missing parts default to NYC placeholders (city New York, state NY, ZIP 10001); for Property we insert the parsed parts as they are (the dataset supplies complete addresses).
- Beds/Baths. parse_bed_bath_info interprets strings such as "3BR/2.5BA" and "Studio/1BA" and returns numeric counts (bedrooms:int, bathrooms:float).
- Client blurb. parse_client_info separates name | profession | Budget … | notes and standardizes K/M suffixes into numeric ranges; it returns name, profession, min/max budget, and notes.
- Appointments. parse_appointment_history expands a bar-separated history into one record per event, mapping phrases like "Final walkthrough" to the inspection enum. The raw sentence is kept in notes.
- Features and documents. Comma lists are split into one row per amenity in PropertyFeature; parse_documents_required maps document names to our enum (e.g., *Title Report* → title_report).
- Category mapping. map_property_type, map_transaction_status (property status), map_transaction_status_enum (transaction status), map_campaign_type, map_lead_source, and map_payout_status align upstream spellings with our enums (for example, "Direct Mail/Direct Marketing" → advertisement, "PAID" → paid).
- Fallbacks. If offer_date is missing we use listing_date or today's date; if offer_amount is missing we fall back to final_price or list_price. All fallbacks are explicit in the log.

4) Load: table-by-table logic and keys

The loader proceeds in dependency order. Each insert checks for an existing row using an indexed business key and then performs an upsert where supported.

- Office. Look up by office_name. If absent, insert_or_get_office creates a deterministic office_code, parses the address, applies NYC defaults when needed, and inserts one row.
- Employee. Deduplicate by email. insert_or_get_employee parses the full name, sets job_title='Agent', and lets the database trigger generate employee_code. An office must exist first (Office→Employee is 1:M).
- Client and ClientRole. insert_or_get_client creates one master row per email (or a stable synthetic email if missing) and immediately adds a role in ClientRole that matches the deal type: buyer/seller for sales; renter/landlord for rentals (Client→ClientRole is 1:M). ON CONFLICT (client_id, role_type) DO NOTHING prevents duplicate role assignments.
- PropertyType. insert_or_get_property_type maps abbreviations (e.g., TH → Townhouse) and ensures a single row per type.
- Property. The loader uses the CSV mls_listing_number as the natural key for Property.mls_number. It then issues an INSERT into PostgreSQL with typed address fields, list price, square footage, bedrooms, bathrooms, and the mapped status; ON CONFLICT (mls_number) updates current_status and list_price to make re-runs idempotent.
- PropertyFeature. After the property exists, the amenity list is split and written one per row (feature_type='amenity'), using ON CONFLICT DO NOTHING where a uniqueness rule is defined.
- Transaction. Keyed by transaction_code (from the CSV column transaction_id). The insert links the property, listing and selling agents, and the parties (buyer/seller or renter/landlord) according to the deal type; status is mapped to our enum. Required dates and amounts use the fallbacks above. A database check enforces that sales cannot carry renter/landlord and rentals cannot carry buyer/seller. ON CONFLICT (transaction_code) updates status and transaction_amount.

Secondary facts (post-commit). After the core commit, child facts are attached in separate transactions—each wrapped in its own try/except with an independent commit/rollback, so a non-critical failure cannot erase the core insert:

- insert_appointments writes event rows for showings and inspections (status='completed', scheduled_datetime at midnight of the event date, created_by = listing agent).
- insert_commission_record records total commission and agent splits. If explicit split amounts are parsed from text, we use them; otherwise, and in the absence of a parsed selling split, the listing agent receives the full amount. ON CONFLICT (transaction_id) DO NOTHING protects the 1:1 relation.
- insert_documents creates required document rows and, when an appraisal amount is present together with an inspection date, also creates an appraisal report entry.

- insert_client_leads converts a buyer blurb and lead source into a structured lead; by rule, interest_type='renting' when budget_max < 50,000, otherwise buying. Leads are inserted with status converted.
- insert_property_media seeds a primary photo and a floor plan for each property.
- For rentals, a Lease is created with derived start/end dates (from lease_start_end or defaulting to closing_date + 365 days) and a lease_number derived from the transaction code, followed by PaymentRecord rows for the security deposit (if present) and the first month's rent.
- MarketingCampaign is inserted when campaign type and spend are present (status completed), using the mapped campaign enum.

5) Validation, deduplication, and idempotency

Lookups hit indexed columns where relevant: office name, employee email, client email, MLS number, and transaction code. Base entities use ON CONFLICT to convert duplicates into updates where business rules allow (for example, Property, Transaction, Commission, ClientRole). Secondary entities that are additive by nature (features, media, documents, appointments, leads, payments, campaigns) are attached after the core commit; repeated loads of the same CSV will append additional rows in those child tables.

6) Transaction boundaries and logging

The base chain (Office → Employee → Client/ClientRole → PropertyType → Property → Transaction) runs inside a single database transaction and commits or rolls back as a unit. Each secondary group (appointments, commissions, documents, leads, media, lease, payments, campaigns) is wrapped in its own transaction with an immediate commit on success and a localized rollback on failure. Logging is at three levels: INFO for progress, WARNING for recoverable data issues and fallbacks, and ERROR for row-level rollbacks with the relevant keys.

7) Runtime notes

The loader performs row-by-row inserts via psycopg2 with explicit commits at safe points. With our current dataset (≈50 rows), a full run completes well under a minute, including normalization and secondary inserts. The job can be re-run safely for base entities because keys, enum mappings, and upserts are consistent across entities.

Analytics Applications

To enable Dream Homes NYC to truly turn data into productivity, during the code implementation process, we translated all queries into scenario-based decision-making actions.

Firstly, the analysis of "what functions can be purchased in different price ranges" was made into an interactive heat map: The agent only needs to drag the slider on the iPad, and the client can see in real time the three most frequently seen supporting facilities at 500,000, 750,000 and 1,000,000. On the spot, the viewing list can be reduced to the five units that best fit the budget, saving the time of back-and-forth communication.

What the marketing team cares about most is whether the money is well spent. With this in mind, we have solidified the lead cost, conversion rate and ROI of each channel in the "Marketing cockpit" of Metabase. At 8 a.m. every day, the dashboard automatically pushes a wech-style summary: Last night, the Instagram AD brought 23 leads, expected to convert 3 orders, with an ROI of 5.4 times. Google keywords only brought in 7 leads, and the ROI dropped to 1.2 times. Based on this, the marketing manager can complete the budget reallocation for the week before 10 a.m., instead of waiting until the end of the month as in the past to find that half of the advertising expenses have been wasted.

For landlords, the most concerning issue is undoubtedly "How much more rent can I earn by installing this set of facilities?" Therefore, we have encapsulated the rental premium calculation into a one-click report: select "gym", "doorman", and "rooftop garden", and the system immediately provides an average premium of 320 US dollars, 280 US dollars for doormen, and 150 US dollars for the gym, and displays the area with the highest premium according to the district heat map. Based on this, the landlord decided to prioritize upgrading the security guard system, expecting to recoup the investment in 14 months, rather than blindly spending money on decoration.

In the past, the timing of selling a house was entirely based on experience. Now, we use the quarterly transaction speed to inform customers of the most favorable time to list. The system will automatically send emails to all potential sellers every March. According to the data of the past two years, houses listed from April to June can be sold on average within 32 days, which is 20 days faster than at the end of the year, and the premium is 3% higher. After receiving the email, the customer only needs to click "Book Valuation", and the schedule will be automatically arranged to the first week of April. The transaction rhythm is completely driven by data.

For the lease structure, we calculated the vacancy rate, rent and deposit for the three models of short-term rental, public rental and long-term rental all at once. The operation team found that although the unit price of short-term rental is 15% higher, the vacancy period is extended by 8%, and the overall income is actually lower. So the background system changed the default recommended contract from 6 months to 12 months. The system automatically

highlighted "Annual rent is more cost-effective" on the tenant signing interface, which not only stabilized the cash flow but also saved the labor of repeated renting.

The security premium of the area that investors care about most has been made into an enlarged map: the darker the color, the higher the proportion of security facilities and the more expensive the housing price. When investors click on any postal code, the average housing price, security coverage rate and the predicted increase in the next 12 months of the area will immediately pop up. With just a few mouse movements, one can identify the value trough of "few security facilities, low housing prices, but a subway under construction", and complete the acquisition before the price increase.

Finally, the lead conversion funnel and the ROI of marketing activities were merged into one "Today's Battle Map". The sales morning meeting turns on the large screen, allowing you to immediately see which channel failed yesterday and which broker had the highest conversion rate. The system automatically redistributes 20% of the fallen leads to the top three brokers of the week, ensuring that every potential customer can be followed up within 24 hours. Data is no longer a monthly report but a metronome that directs business in real time.

.

Metabase Dashboard

Conclusion

In this task, we helped Dream Homes NYC transform the initially scattered old forms and scanned copies into an online data warehouse that can be answered at any time. The database itself follows the three normal forms, eliminating redundancy. Foreign keys and constraints ensure that every record is trustworthy. The Python-automated ETL operates silently every night, flowing new properties, customers, transactions, and payments into the corresponding tables. If an error occurs, it rolls back, ensuring that the report appears on time the next day even when unattended. Twelve business-level queries cover the most frequently asked questions by brokers, marketers, renters, and investors. Metabase hides these technical details behind the scenes, allowing any role to use clicks instead of code.

Finally, we have enabled brokers to take out their mobile phones at the viewing site and check the average transaction price of the same community last month within two seconds. The

marketing manager can directly pull out the ROI of each lead in the budget meeting and immediately cut off inefficient channels. Executives can open the dashboard to see the total revenue for the quarter, occupancy rate, and which office has the fastest transaction rate. Decision-making has been shortened from weeks to hours. More importantly, this architecture is horizontally scalable on the cloud: doubling the number of listings and tenfold the concurrent queries, all that is needed is to add nodes, without touching a single line of business code. Dream Homes NYC now has a truly "data heart" that grows along with the business. In the future, whether it is to integrate smart home IoT data, conduct machine learning valuation, or launch a customer self-service house-finding mini-program, it can smoothly grow on the existing schema without having to start from scratch.