COURSE NAME: Database Programming

FINAL PROJECT ASSIGNMENT

COURSE CODE: COSC-2307

STUDENT NAME: Deny Setiawan Rahardjo (Jojo)

STUDENT ID: 5149978

INSTRUCTOR: Prof. Liam Dingle

FINAL PROJECT REPORT

1. Project Introduction

The Room Rental Management System (RRMS) is a fully normalized relational database designed to support multi-landlord rental operations, recurring rent billing, payment tracking, and financial reporting. The system provides a clean separation of master data (landlords, properties, units, tenants, contracts), transactional data (rent charges, payments), and reference data (billing cycles, statuses, penalty rules, assets, utilities).

The primary objective of this project is to model and implement an operationally accurate and analytics-ready database foundation for a future SaaS rental management platform. The design ensures:

- multi-landlord isolation through hierarchical scoping
- strong referential integrity across rental operations
- BCNF-level normalization
- support for recurring billing logic
- time-based reporting, such as revenue, occupancy, and overdue balances

The RRMS database was implemented in PostgreSQL, populated with sample data, and connected to Power BI to demonstrate operational and analytical capabilities.

2. Business Problem / Use Case

Small landlords and independent rental operators often manage their business using spreadsheets or manual notes. These fragmented processes cause problems such as:

- missing or inconsistent rental records
- no automated billing reminders
- difficulty tracking overdue charges

- no consolidated financial reporting
- difficulty maintaining historical contract information

To address these issues, the RRMS database focuses **exclusively on billing and payment tracking**, beginning *after* a landlord and tenant already have an agreed-upon rental relationship.

RRMS Use Case Summary (Aligned with Design Documentation V2)

The database supports:

➢ **Multi-Landlord Operations**

- Each landlord maintains an independent portfolio of properties, units, tenants, contracts, and financial data.

➢ **Property & Unit Management**

- A landlord owns multiple properties.
- A property contains one or more units (rooms, apartments, suites).

➢ **Tenant & Contract Management (Billing-Centric)**

- Store tenant identity and contact information.
- Store contract terms: rent amount, start/end dates, billing cycle, billing day, penalty rule, and status.
- Maintain historical and active contracts without complex approval workflows.

➢ **Recurring Billing Engine**

- Automatically generate rent charges per billing cycle.
- Track due dates, penalties, and outstanding balances.
- Support partial or multiple payments per rent charge.

➢ **Asset & Utility Listing**

- Store simple lists of furniture, appliances, and utilities for each unit.

- **Reporting & Analytics**
  - Monthly revenue
  - Occupancy rate
  - Overdue payments
  - Active vs historical contracts
  - Per-landlord performance
  - Annual revenue projection

This use case aligns directly with the conceptual & logical model described in the Design Document V2.

3. Scope & Requirements

3.1 Project Scope

The project includes:

- designing the conceptual, logical, and physical schema
- normalization to BCNF
- implementation in PostgreSQL
- loading sample data
- executing operational SQL queries
- building Power BI dashboards

The RRMS database focuses only on billing-related operations:

Landlord → Property → Unit → Contract → RentCharge → Payment

Other functionality (maintenance, messaging, or workflow approval) is explicitly out of scope.

3.2 Functional Requirements

A. Multi-Landlord & Property Management

- Store multiple landlords.
- Each landlord owns multiple properties.
- Each property contains one or more units.

B. Tenant & Contract Management

- Store tenant profiles.
- Each contract links a tenant to a specific unit.
- Contract includes:
    - start_date, end_date
    - base_rent_amount
    - billing_day (due day)
    - billing_cycle_type
    - penalty_rule
    - contract_status
- Only one active contract per unit at a time (business rule).

C. Asset & Utility Listing

- Store furniture and appliance records per unit.
- Store utilities/services (hydro, water, internet) per unit.

D. Rent Charges, Penalties & Payments

- Generate periodic rent charges.
- Track charge statuses (Pending, Paid, PartiallyPaid, Overdue).

- Support multiple or partial payments per charge.
- Compute penalties based on penalty rules.
- Provide complete rent and payment history.

E. Overdue & Outstanding Balances

- Identify overdue charges based on due dates.
- Provide outstanding balances for tenants, units, contracts, and landlords.

F. Reporting & Analytics

- Monthly and yearly revenue
- Occupancy rate
- Overdue trends
- Portfolio performance
- Contract history
- BI-ready structure

3.3 Non-Functional Requirements

- BCNF normalization
- Strong FK and domain constraints
- Indexing on FK columns and date fields
- Multi-landlord isolation
- Analytics-friendly schema
- Scalability for historical data growth

4. Architecture Overview

4.1 System Architecture (Aligned with V2)

The RRMS architecture consists of:

1. Client Layer

- Landlords: manage units, contracts, charges, and payments.
- Tenants: receive receipts (future extension).

2. Application Layer (Future)

(Not implemented in this project)

- Forms for properties, units, contracts
- Receipt generator
- Notification engine
- API layer

3. Database Layer (Implemented)

Contains the full schema:

Landlord → Property → Unit → Contract → RentCharge → Payment
+
Lookup/reference tables: BillingCycleType, ContractStatus, ChargeStatus, PaymentMethod
+
Supporting tables: PenaltyRule, UnitAsset, UnitUtility

This design directly mirrors the conceptual model from the Design Documentation V2 .

4.2 Logical Data Flow

1. Landlord creates property and units
2. Tenant data added
3. Contract created with billing/penalty rules
4. System generates recurring rent charges
5. Landlord records payments

6. System updates charge statuses
7. BI dashboard reports monthly revenue, occupancy, overdue, etc.

4.3 Multi-Landlord Isolation

Hierarchical scoping ensures isolation:

Landlord → Property → Unit → Contract → Charge → Payment

All financial and operational data trace back to landlord_id indirectly.

4.4 Technology Stack

- Database: PostgreSQL
- Design: Draw.io
- SQL Tools: pgAdmin / VS Code
- Analytics: Power BI
- Version Control: GitHub

4.5 Extensibility

Future enhancements may include:

- Tenant portal
- Automated email billing
- Payment integrations
- Subscription billing for landlords
- Maintenance module
- Mobile/web frontend

5. Design Documentation Summary

The design of the Room Rental Management System (RRMS) database follows a structured top-down methodology, beginning with the conceptual design and progressing through the logical and physical layers. The purpose of this design process is to ensure strong data consistency, BCNF-level normalization, multi-landlord isolation, and analytics-ready data structures.

The conceptual model identifies the core entities required for billing-centric rental operations, including Landlord, Property, Unit, Tenant, Contract, RentCharge, and Payment. Supporting lookup entities such as BillingCycleType, ContractStatus, ChargeStatus, and PaymentMethod standardize operational workflows. Additional entities such as PenaltyRule, UnitAsset, and UnitUtility provide operational metadata used for reporting and rental administration.

The logical model formalizes these relationships using PK-FK constraints, enforcing business rules such as:

- one landlord owns multiple properties;
- one property contains multiple units;
- one contract generates multiple rent charges;
- one rent charge may have multiple payments;
- penalty rules, billing cycles, and statuses may apply to multiple contracts or charges.

The physical implementation in PostgreSQL uses identity primary keys, strict foreign key constraints, check constraints, and indexes on high-usage columns (e.g., landlord_id, unit_id, contract_id, rent_charge_id). This ensures efficient performance for both transaction operations and analytical workloads.

The complete design documentation, including the full conceptual model, logical ERD, physical schema, normalization notes, and entity descriptions, is available in the dedicated GitHub repository.

Full design documentation is available at:
GitHub – RRMS Database Design https://github.com/Setya78/Final-Project-COSC2307/tree/main/docs

6. PostgreSQL Implementation

The implementation of the Room Rental Management System (RRMS) used PostgreSQL 17, with all schema creation, indexing, data seeding, and demonstration queries organized in a dedicated SQL folder. This structure ensures reproducibility, clarity, and maintainability throughout the development process.

All PostgreSQL operations were executed directly using the SQL scripts described below.

SQL Folder Overview

This folder contains all PostgreSQL scripts required to build and populate the RRMS database:

1. schema_rrms.sql

Defines the complete database schema, including:

- creation of all entities described in the logical model
- primary keys using GENERATED ALWAYS AS IDENTITY
- foreign key relationships ensuring referential integrity
- CHECK constraints for business rules such as billing_day and penalty_type
- NOT NULL constraints for required fields

This script provides the full relational design for the RRMS dataset.

2. index_rrms.sql

Contains all indexing commands used to optimize database performance.
Indexes are applied to high-frequency join and filter columns, including:

- landlord_id → property
- property_id → unit
- unit_id → contract
- contract_id → rentcharge
- rent_charge_id → payment
- status fields used in reporting queries

These indexes significantly improve analytical queries such as revenue calculation, occupancy rate monitoring, and overdue detection.

3. seed_rrms.sql

Provides a structured sample dataset used for validation, testing, and Power BI reporting.
This script includes initial records for:

- landlords, properties, and units
- tenants and contracts
- penalty rules, billing cycle types, and statuses
- sample rent charges
- sample payments (including partial payments)

The seeding script contains **2–3 sample rows per table**, ensuring the system can demonstrate full end-to-end workflow without overwhelming the database.

4. sample_query.sql

Contains demonstration queries used in:

- the PostgreSQL video walkthrough
- the reporting section of the project
- Power BI measure validation

Queries include:

- monthly revenue summaries
- occupancy rate calculations
- overdue detection logic
- outstanding balance per tenant
- contract and payment history views

This file shows how the schema supports real operational and analytical use cases.

Repository Link

All SQL scripts described above can be accessed at:

GitHub – PostgreSQL Implementation Scripts
https://github.com/Setya78/Final-Project-COSC2307/tree/main/SQL

GoogleDrive– SQL Query Video

https://drive.google.com/file/d/1cHgAAKU9aHI593L_aT4VPAj8mCGXYikY/view?usp=drive_link

7. Data Loading & Sample Datasets

A complete set of dummy datasets was prepared to populate the RRMS database and support testing, reporting, and demonstration. All datasets are stored in the **/data** folder of the GitHub repository and provided in CSV format.

The folder includes sample records for all major entities, such as **landlords, properties, units, tenants, contracts, rent charges, payments, penalty rules, billing cycles, and utilities**. These files represent realistic rental scenarios and allow the system to demonstrate key features such as contract linking, recurring billing, overdue detection, and revenue reporting.

The datasets serve three main purposes:

- **Populate the PostgreSQL database** with meaningful test data
- **Support SQL demonstrations**, including JOINs, GROUP BY, HAVING, and analytical queries
- **Act as the data source for Power BI dashboards**, enabling revenue, occupancy, and overdue insights

All sample datasets can be accessed at:

GitHub – RRMS Data Folder https://github.com/Setya78/Final-Project-COSC2307/tree/main/data

8. Power BI Reports

The Power BI dashboard developed for this project provides a visual summary of key rental, payment, and occupancy metrics for the Room Rental Management System (RRMS). The report is directly connected to the PostgreSQL database and reflects real data from the RRMS schema. The dashboard includes high-level indicators such as occupancy rate, outstanding balances, total rent generated, monthly payment breakdown, and payment method distribution. These visuals demonstrate how the RRMS database supports business insights through analytical reporting.

All Power BI files, DAX measures, and documentation, including the full dashboard video walkthrough, are stored in the project repository.

Github – Power BI Files https://github.com/Setya78/Final-Project-COSC2307/tree/main/powerbi
Google Drive – Power BI Dashboard Video Documentation
https://drive.google.com/file/d/1sVKi82JPnz9FKmma0B1IB9KRJKH5Cdc6/view?usp=drive_link

9. Lesson Learned

Throughout the development of the Room Rental Management System (RRMS) database, several important lessons emerged regarding database design, system requirements, and analytical reporting.

1. **Clear separation of conceptual, logical, and physical design is essential.** Starting with the conceptual model enabled a stable understanding of business rules before committing to SQL implementation. This prevented rework and ensured referential integrity.
2. **Normalization and lookup structuring greatly reduce long-term complexity.** Moving domain-specific values such as status codes, penalty rules, and billing cycles into dedicated lookup tables simplified updates and improved data consistency.
3. **Hierarchical multi-landlord scoping must be planned early.** Designing the Landlord → Property → Unit → Contract → Charge → Payment chain from the beginning made analytics significantly easier and ensured natural data isolation.
4. **Recurring billing logic is easier to implement when the database is clean.** By modeling contracts as the "source of truth," monthly charge generation became straightforward and predictable.
5. **BI tools require a schema that is query-friendly.** Indexing foreign keys, using consistent surrogate keys, and storing time-based attributes (charge_period, contract start/end dates) made Power BI reports fast and reliable.
6. **PostgreSQL constraints save the system from silent data corruption.** CHECK constraints, foreign keys, unique indexes, and NOT NULL rules helped guarantee that only valid and meaningful data can enter the system.

7. **Version control and documentation are critical.**
   Using GitHub ensured traceability, reproducibility, and a professional workflow similar to real industry projects.

These lessons reinforce that a well-designed relational database is not only an operational tool but also a foundation for analytics, reporting, and future SaaS scalability.

10.

11. Conclusion

The Room Rental Management System (RRMS) project successfully delivered a fully normalized, operationally accurate, and analytics-ready relational database for multi-landlord rental management. The design supports the entire billing cycle, from unit and contract setup to automatic rent charge generation, payment tracking, overdue identification, and monthly revenue reporting.

By following a structured development lifecycle (conceptual → logical → physical), the final database meets all functional and non-functional requirements, including BCNF normalization, strong referential integrity, multi-landlord isolation, and scalable analytical performance.

PostgreSQL implementation, sample datasets, and Power BI dashboards demonstrate that the RRMS schema is ready for future SaaS extension, including automation, tenant-facing portals, and integration with online payment systems.

Overall, this project provides a solid and future-proof foundation for a commercial-grade rental management solution.

All supporting materials for this project are available in :

a. ***GitHub repository:*** ***https://github.com/Setya78/Final-Project-COSC2307/tree/main***
b. ***GDrive:***
   https://drive.google.com/drive/folders/16Mfmxg7ZN2BH0xMbppOktuTO5ligXh0_?usp=drive_link