Technical Report

Rene Gliddon

University of Maryland Global Campus

DBST 651: Relational Database Systems

Luis Gonzalez

Date: November 01, 2022

Table of Contents

1. Introduction

This report describes the design and implementation of a database that will store data for a rental property management application. The database will store data related to the users of the database, the rental properties, the tenants, all expenses and incomes incurred from the properties, as well as the transactions on the database itself. The purposes of this rental property application database are to assist property owners with managing their business data, making good decisions, and seeing the larger picture. This database will form the heart of the rental property application-storing the raw data and producing the information requested from the user to meet the purposes listed above.

2. Overview

This technical report will cover the design of the database only and does not include the design of the rental property application itself. This technical report includes a literature review, assumptions made, design decisions, Statement of Work, Requirements Definition Document, detailed database design information (ERD, DDL, DML, sample queries), as well as a final section on database administration and monitoring.

The database will interact directly with an application server that will receive the data input by the user. Therefore, the database will interact indirectly with the user. The user must be an authorized user (property owner has the right to authorize users), and the level of functionality available to the authorized user is dependent on the level of rights the property owner has set for that user. These rights are either “Owner” level (read, write, and edit all data), or “Tenant” level (read only selected data).

The first goal of the database is to securely store all data relating to the property owner’s properties, tenants, expenses, incomes, as well as transactions on the database. The second goal is to manipulate the data to show meaningful information to the property owner and any authorized

property manager(s) in order to see a larger picture of the trends, extract helpful information, and make good decisions. The third goal of the database is to show limited information to tenants. This information includes limited information regarding the tenant in question's own personal information and rental contract(s). Overall, the database should assist the property owner in managing the properties and maximizing profit.

3. Literature Review

A related database that was used in rental price and returns analysis was used in the article "Measuring Local Individual Housing Returns from a Large Transaction Database" (Gregoir et al, 2012). This article discusses a large database that was created using data from rental properties in Paris. The purpose was to calculate the return on investment in the Parisian housing market. Lease contracts, supply and demand, vacancy, risk, rental returns and capital gains, averages and variances are discussed, among other things. The article also discusses the above factors based on the size of the property, number of rooms, and locations.

The database discussed in this report can be used for a number of financial calculations (average rent in a given zip code or state, outliers, return on investment, rent to price rations, etc.) including other calculations involving vacancy times and locations, among other things. The difference between the database in the article, and the one being discussed in this report are in its users and purposes. The data from the article was submitted by property owners, and the authors calculated the results for publishing purposes. The database in this report will have its data submitted possibly by property manager(s), as well the property owner, and the information calculated by the database will be used by the property owner (and perhaps property managers as well) to achieve the goals outlined-ultimately to maximize return on investment and to highlight areas of problems and opportunity.

4. Assumptions

- It is assumed that the property owners and managers are using either a laptop or desktop computer to use the rental property app, and hence the database.
- It is assumed that the device used to access the database has approximately 2GB of RAM available.
- It is assumed that the property owner and/or property manager(s) will keep the database up to date by entering in each new rental contract.
- It is assumed that the property owner and/or property manager(s) will keep the database up to date by marking a rental contract as not current once it has passed and a new rental contract has been entered into.
- It is assumed that the property owner and/or property manager(s) will keep the database up to date by entering in each new expense and income.
- It is assumed that the property owner and/or property manager(s) will keep the database up to date by entering in each new tenant, as well as updating any information regarding the tenant that may have changed (contact details, etc.).
- It is assumed that the property owner and/or property manager(s) will keep the database up to date by entering in each new property, as well as updating any information regarding the properties.

5. Design Decisions

5.1 Key Factors Influencing Design

The design of the database was initially modelled as an Entity Relationship Diagram. The design of the database was modeled in an object-oriented manner, with the main entities representing different objects in the property management business: users of the database that have attributes (“user_code”,

security level, username, password, etc.), properties that have attributes (location, costs, etc.), tenants that have attributes (name, contact information, other information specific to each tenant that can be listed under “tenant_notes”, etc.), rental contracts that are agreed upon between the property owner/manager(s) and the tenant that have attributes (rent amount, dates of validity, property, tenant, etc.), expenses incurred that have attributes (property, date, reason, description, amount, etc.), incomes received that have attributes (property, date, reason, description, etc.), transactions on the database that have attributes (date, time, transaction type, etc.). The idea of each entity being an object with attributes that interacts with the other objects based on a common attribute, or key, is the idea behind the design of this database. In property management, each entity interacts with others based on common factors.

5.2 Functional Design Decisions

The functional design decisions of this DBMS will center around the application server and user interface. The user will input or update data into the database by inputting it through the UI in the appropriate place. This will then connect to the application server, which connects to the database. A user will also log in through the UI, and thus the application server and database, and request to view information through the UI, then application server, and finally database. The database will send information to the UI via the application server as well. The UI should be intuitive and be able to run on Windows, Linux and MacOS. Once the user has logged in, there should be options to view information, update data, and add new data to the database, depending on the user’s security clearance level.

5.3 Database Management System Decisions

The DBMS chosen is Oracle as it is a well-known DBMS with high availability and performance (Nguyen, 2022). In addition, the pricing is scalable, which suits the business of a property owner managing their properties, as a property owner can expect to start small and grow their business over time as they gain more capital for further investment in rental properties. Furthermore, Oracle is secure and offers data encryption and multi-factor authorization, and IP blocklists. There is also a Recovery Manager (RMAN) available to recover and restore data.

5.4 Security and Privacy Design Decisions

The database will require users to be pre-authorized with a code issued by the property owner. The property owner can create a new user in the database and assign the user a security level of either “Tenant” or “Owner”. The database will assign the user a user code (“user_code”). The user can then complete their registration by choosing a username and password which will then be entered into the database and used to log in. The security levels will allow the users to only be able to perform the functions that the property owner wants them to be able to perform, and only be able to see the data/information that the property owner wants them to be able to see. In addition, this protects the other tenants’ privacy, as tenants can only see a limited view of their own data.

6. Statement of Work (SOW)

1.Overview

This project develops a database for use in a real estate app for landlords and tenants. The objectives of the database are to store data relating to tenants, properties, rental contracts, expenses, incomes, and users, as well as to display and manipulate that data. The data will be both quantitative and qualitative. Both data types will be used to inform, make good decisions, record, and analyze. The quantitative data can be used to determine above average expenses per property or tenant in a given zip code, return on investment, abnormalities, and other such analyses. The qualitative data can be used to create a list of tenants who require a renewed rental contract, a list of tenants who are behind in their ren to inform, record data, as well as be used together with the quantitative data to make decisions, providing context for the financial data.

2. Objectives

The purpose of this project is to create a database for landlords and their tenants to assist with managing the properties and tenants, keep records, allow convenient access to view information, secure information, and to perform searches and calculations on demand.

The database will enable the creation and maintenance of a repository to hold all information relating to the landlord's property management, including information regarding properties, tenants, rental contracts, any expenses or incomes incurred, as well as a table for database users' information (username, password, security level). The database will store information for record keeping and reminders; calculate expenses, incomes, break-even points, profit, and loss; show tenants that are causing more expenses than the average in their zip code, tenants that require updated rental contracts, tenants that are behind on their rent; help landlords make decisions; and allow tenants on-demand access to their information.

Users will be able to log in (or sign up with a code from the landlord), and based on their security clearance, will be able to perform limited searches from the database. Tenants will have limited viewing rights to their own details, their contract details, and the property details. The security clearance for a tenant will allow the tenant to submit a change request for any data point relating to their tenant information, or rental contract information, as well as to view most of their rental contract details, property information, and tenant details (apart from the landlord's notes on the tenant, rental contract, and property, as well as any financial information besides rent owed by the tenant). The landlord will have total viewing, editing, adding, and removing rights. The landlord will be able to view all property, tenant, rental contract, expense, income, and user details. The landlord will be able to add new users, properties, rental contracts, tenants, expenses, and incomes. The landlord will also be able to remove any of the above and edit any of the above. The landlord will be able to give a tenant a user code that comes with a security level (tenant or landlord). Users will be able to sign up using this code- creating a username and password to log into the system. The application will generate and verify the code.

The database creation project will consist of the following objectives and deliverable deadlines:

1. Create and submit a Statement of Work by Tuesday, September 13, 2022
    a. Include objectives, tasks, deliverables, and constraints
    b. Define purpose and scope
    c. Define benefits
2. Analyze business domain and define requirements in a Requirements Definition Document by Tuesday, September 27, 2022
    a. Identify business data
    b. Identify how business data is related to each other

   c. Describe business rules by describing entities and attributes, primary keys, foreign keys, relationships between entities, cardinalities, and any special considerations

3. Design Database Model via an ERD by Tuesday, September 27, 2022
   a. Create initial conceptual level ERD, consisting of 5 – 6 entities, each with at least 5 attributes (including primary keys, excluding foreign keys)
   b. Normalize ERD design, eliminating M:N (many to many) relationships
   c. Design using Crow's Foot notation, ensuring relationships are labelled and special considerations/assumptions are commented
   d. Design using ER Assistant
   e. Ensure primary and foreign keys are present in design
4. Create Data Definition Language (DDL) SQL Script by Tuesday, October 11, 2022
   a. Implement database using DDL
   b. Create tables, columns, keys, indexes, views, and triggers based on business rules and including data types, size and any constraints
   c. Create parent tables first, then child tables, adding columns and names
   d. Add any indexes required
   e. Add audit columns
   f. Add table views where audit columns are not seen, as well as any other views needed
   g. Create sequences to generate primary keys
   h. Add triggers to automatically generate primary keys
   i. Populate tables with business data'
   j. Query tables

5. Solve business problems and provide insight with SQL statements by Tuesday, October 11, 2022
6. Deliver SQL script as well as a technical report by Tuesday, November 1, 2022

3. Project Scope

The database will form part of an app to help manage real estate data, however the scope of this project is limited to the design and development of the database that will feed data to the app for viewing, calculation, updating and deletion purposes. In scope work will include documentation the project objectives, deliverables, constraints, purpose, and requirements; defining business rules; designing and modelling the database via an ERD, creating a DML SQL script, implementing the database, testing the database with queries, and delivering a full technical report.

3.1 Work within the scope of this project

- Statement of Work, including objectives, tasks, deliverables, constraints, and scope
- Requirements Definition Document, including business data and rules
- Entity Relationship Diagram modelling the database
- Data Definition Language SQL script
- Consolidated DDL/DML script
- Final report

3.2 Work outside of the scope of this project

- The actual app that will utilize the database

4. Database Goals, Expectations, and Deliverables

The database goals and expectations include designing and developing a database that consists of 7 tables, with columns listing at least 5 attributes per table, each table with a primary key and

possibly a foreign key where applicable. Each table should consist of data points regarding the entity, a unique index where applicable, database objects (such as sequences, views, and triggers) that will facilitate data entries and queries. There should also be sample SQL database queries. The deliverables include this Statement of Work (by Tuesday, September 13, 2022), Requirements Document (by Tuesday, September 27, 2022), ERD (by Tuesday, September 27, 2022), DDL script in plain language (by Tuesday, October 11, 2022), Consolidated DDL/DML script (by Tuesday, October 11, 2022), final project report (by Tuesday, November 1, 2022).

5. Database Benefits

The financial benefits of this database include quick retrieval of financial details, return on investment analysis, analysis of properties that drain funds and properties that provide positive financial value with the relevant tenants and contracts. The informational benefits include keeping a record of all tenants, property details, and rental contract details; as well as showing which tenants have rental contracts that are expired and require renewing (and/or tenants that rental contracts are close to expiring and require a renewed rental contract), which tenants are behind on rent, etc. The non-quantitative (non-financial/non-numerical) data has benefits such as reducing the time taken to retrieve information, and increasing the security of information; but even more than that, the non-quantitative data can be linked with the financial data to help landlords make decisions regarding tenants, rental contracts and properties, such as whether a property should be sold, whether a tenant's lease should be renewed, whether a property should be renovated and leased at a higher price, etc. The database will provide convenience and lead to good financial decisions-ultimately assisting the landlord to increase the profit of his/her real estate investments.

6. Hardware and Software (10)

a) Diagramming Tool

ER-Assistant Version 2.10 running on Windows 11 Pro

b) Diagram and Access Method Identified

UMUG VDA connection with Intel(R) Xeon(R) CPU E5-2673 v4 @ 2.30GHz 2.29 GHz processor, running on the Windows 11 Pro Version 21H2 operating system. This 64-bit operating system has 32GB of RAM.

c) Office Productivity Tools

Microsoft Office 365

d) Client Hardware

A regular computer running Windows, macOS, or Linux with approximately 2GB of RAM.

e) Server hardware (adapted from "Server Hardware Sizing Recommendations", 2019, available at https://support.jetglobal.com/hc/en-us/articles/219401657-Server-Hardware-Sizing-Recommendations)

- High speed disk drives with separate storage for SQL log, data, and temp files (+-100GB of storage)
- Processor with +-8 cores
- 16 – 64 GB RAM

f) Development Tools

Oracle SQL Developer Version 21.2.1.204.1703

g) The DBMS system

Oracle Database

7. SQL Usage and Style Guide (5)

Data Definition Language (DDL) will be used to create the database, using CREATE TABLE table_name (column_name DATATYPE size CONSTRAINTS). Parent tables will be created first, and then the child tables. DDL will also be used to create any necessary indexes, sequences, views and triggers to the tables. Once we have defined the database, we will enter data into the tables via Data Manipulation Language (DML). This will be done via INSERT statements (INSERT into table_name(column1, column2…)VALUES(value 1,value2…); COMMIT;). Basic queries will also be done using DML such as SELECT statements, delete statements, update statements, as well as joins, sub-queries, etc.

SQL Statement Structure for Readability

- Upper case will be used for SQL commands, keywords, data types, functions
- Lower case will be used for column, table, and variable names
- White space will be used on both sides of "="
- Various clauses of a statement will be on separate lines
- For DML statements, columns will be listed on a separate line

Script Format

- Scripts will be in plain text
- Scripts will be in a TXT or SQL file
- Script will be saved as LastName_Firstname_DDL_DML.txt, LastName_Firstname_DDL_DML.sql, LastName_Firstname_DDL.txt and/or LastName_Firstname_DDL.sql

Comment Usage

- Comments will be opened with “/*” and closed with “/* “, or begun with “- -"

Object Naming Conventions

- Names (tables and columns) must be unique within the database schema.
- Names should be as short as possible, and not longer than 30 characters
- Names should not include spaces or hyphens, but can include underscores
- Names must begin with a letter
- Names should be singular
- Names should describe the nature of the data contained in the table

7. Requirements Definition Document

Entity and Attribute Description

This section outlines the entities, their attributes, as well as specifying their primary and foreign keys.

1. Entity: PROPERTY

   The entity PROPERTY will describe the property that the property owner owns. The table will keep all basic information about the PROPERTY. The primary key for this entity is the PROPERTY's nickname (property_nickname). There is no foreign key.

   Attributes:

   - property_code: This is a NUMERIC, database generated code to identify the property.
   - property_nickname: This is the VARCHAR nickname of the property that the property owner will choose. It must be unique and meaningful. This is the primary key and can't be null.
   - mortgage_years: This is the number of years that the mortgage will need to be paid until the property has been paid in full. It is of a DECIMAL data type. It can't be null but can be set to 0.
   - property_address: This is the VARCHAR address of the property and can't be null.
   - state: This is the VARCHAR 2 letter code of the state that the property is in.
   - zipcode: This is the VARCHAR zipcode of the area that the property is in.
   - monthly_mortgage: This is the amount that the property owner needs to pay each month for the mortgage payment, stored in a DECIMAL data format. It can't be null but can be set to $0.00 if there is no mortgage.
   - hoa: This is the Home Owners Association fee, stored in a DECIMAL data type. There may not be a fee, in which case it can be set to $0.00. It can't be left null, however.

- date_bought: This is the date that the property was bought. It is stored in a DATE format.
- sewage: This is the VARCHAR name of the company that deals with the sewage of the property. This is the company that sewage fees need to be paid to.
- electric: This is the VARCHAR name of the company that provides electricity to the property. This is the company to which the electric bills need to be paid.
- property_notes: This is where the property owner can write any notes about the property. These may include renovations that need to be done, or areas of concern or interest to the property owner. This attribute is stored as a VARCHAR data type.

2. Entity: PRIMARY_TENANT

This table will hold the basic information regarding the primary tenant. The primary key is the unique tenant_code, which can be automatically generated by the database or manually created. There is no foreign key.

Attributes:

- tenant_code: This is a NUMERIC code generated for each tenant by the database (or manually entered). It must be unique and cannot be null as it is the primary key.
- tenant_fn: This is the tenant's first name, saved in a VARCHAR format. It can't be null.
- tenant_ln: This is the tenant's last name, saved in a VARCHAR format. It can't be null.
- tenant_email: This is the email address used to contact the tenant, saved in a VARCHAR format.
- tenant_phone: This is the phone number used to contact the tenant, saved in a VARCHAR format. It can't be null.

- tenant_notes: These are any notes regarding the tenant that the property owner sees fit to include. It could possibly hold sensitive information, such as relevant financial or familial circumstances and should be kept private. This is stored in a VARCHAR format and can be left null.

3. Entity: RENTAL_CONTRACT

This is the table that will hold information regarding the terms of the agreement between the property owner and the tenant. The primary key is an automatically or manually generated Occupancy_Code. There are two foreign keys: fk_tenant_code and fk_property_nickname.

Attributes:

- occupancy_code: This is a NUMERIC code that uniquely identifies each rental contract. This attribute can't be left null and must be unique. This is the primary key of this table.
- fk_tenant_code: This is one of the two foreign keys. It references the NUMERIC tenant code of the primary tenant (tenant_code attribute of the table PRIMARY_TENANT) that signed the rental contract. It can't be null.
- fk_property_nickname: This is the second and last of the foreign keys. It references the VARCHAR nickname of the property (property_nickname attribute of the table PROPERTY) with which the tenant has entered a rental agreement via this rental contract. This can't be left null.
- rent: This states the DECIMAL dollar amount (includes two decimal spaces) of the rent that is due each month by the listed tenant for renting the listed property. It can possibly be listed as $0.00 but can't be null.

- charge_electric_to: This VARCHAR attribute states whether the “Tenant” or “Owner” will be receiving and paying the electric bill.
- charge_sewage_to: This VARCHAR attribute states whether the “Tenant” or “Owner” will be receiving and paying the electric bill.
- current_tenant: This is a single character (CHAR) entry of either “Y” or “N” that indicates whether the listed tenant is the current tenant of the listed property. If this field is “Y”, then this is also the current and active rental contract for the listed property. This field can’t be null. Each time a new rental contract is entered into the database with this attributed listed as “Y”, the system code will update all other current_tenant attributes at the listed property in other rental_contract tuples as “N”. There can only be one current primary tenant per property. This will be done by code in the system, and not by the database itself.
- date_start: This is the date upon which this rental contract became active. It will be saved in a DD_MON_YYYY (DATE data type) format. This field should not be left null
- date_end: This is the date upon which this rental contract becomes void. It will be saved in a DD_MON_YYYY format (DATE data type). This field should not be left null.

4. Entity: EXPENSE

This entity lists the details of all expenses incurred by a property. The primary key of this table is an automatically or manually generated expense_code that uniquely identifies this expense at this property. The foreign key of this table is the fk_property_nickname attribute, which lists the property that incurred the expense. Neither the expense_code nor the fk_property_nickname fields may be left blank.

Attributes:

- expense_code: This is a unique NUMERIC code to identify this particular expense at this particular property. It is an automatically generated code that is generated by the database. This is the primary key and must be unique. It can't be null.
- fk_property_nickname: This is the VARCHAR nickname of the property that incurred this expense. It is the foreign key and can't be null.
- expense_amount: This is the DECIMAL dollar value (includes two decimal spaces) of the incurred expense. It can't be null, although it is possible to enter a value of $0.00 (although this would not be considered an expense and therefore is not recommended).
- expense_description: This is the VARCHAR description of the expense incurred. This can't be null.
- expense_date: This lists the date upon which the expense was incurred. This is a DATE data type in a DD_MON_YYYY format. This field can't be null.
- expense_notes: This is to note any unusual circumstances regarding the expense, possible future consideration, etc. This is of the VARCHAR data type and can be left null.

5. Entity: INCOME

This entity describes incomes generated by each property. The primary key is a unique, alphanumeric income_code, which must be unique and can't be null. The foreign key of this table is the fk_property_nickname of the property that generated this income or is associated with this income. The foreign key can't be null.

Attributes:

- income_code: This is a unique NUMERIC code that identifies this particular income at the listed property. It can be autogenerated or manually entered. If auto-generated, the code is generated by the database. This field can't be null.
- fk_property_nickname: This refers to the VARCHAR property_nickname attribute of the property that generated the income (property_nickname attribute of the PROPERTY table). This is the foreign key and can't be left null.
- income_amount: This is the DECIMAL dollar value (includes two decimal spaces for cents) of the income that was generated. It can't be left null, but if necessary, can be set to $0.00, although this is not recommended as it is not a monetary income in that case.
- income_description: This is the VARCHAR description of the income that was received. This can't be null.
- income_date: This is the date that the income was received. It is in a "DD-MON-YYYY" format (DATE data type). It can't be null.
- income_notes: This is to note any unusual circumstances regarding the income, possible future consideration, if it goes to any third party, etc. This is of the VARCHAR data type and can be left null.

6. Entity: USER_LIST

This entity lists the information regarding authorized users for this system. When the property owner wants to allocate another user to the system, he/she creates a unique user code (or allows the database to generate one) with a security clearance level. The user code (user_code) is the primary key. When the user clicks "Sign Up" on the user interface, he/she will be prompted to choose a unique username ("username") and a password ("user_password"), as well as enter an email address ("user_email"). These will be added to the tuple. The

user_password and username fields are entered at the “Login” page of the system. If they both match entries in a single tuple, then access is granted to the degree of the security level. Users of the “Tenant” security level will only be able to view a portion of their own information (Rental_Contract, Primary_Tenant, and Property). There is no foreign key.

Attributes:

- user_code: This is the unique, NUMERIC access code generated by the database to allow another user access to the system. When this tuple is initially created, a user code (user_code) and security clearance level (security_level) are input into the tuple. This user_code can’t be null and must be unique. If the property owner does not input a user_code, then the database will generate the User_Code.
- security_level: This is a VARCHAR attribute describing the level of access the user has to the system. There are two possibilities: “Owner” or “Tenant”. This can’t be null.
- username: This is the unique varchar name chosen by the user to identify him/herself. In order to gain access to the system, this must be in place. However, the tuple can exist with this field null.
- user_password: This is the VARCHAR password chosen by the user to match with the unique username to allow access to the system. This field can be null, but the user won’t be able to gain access to the system.
- user_email: This is the email address of the user and is a VARCHAR data type. This email address can be used by the system for such functions as resetting a forgotten password. This field can be null, but access to the system won’t be granted.

Entity: DATABASE_TRANSACTION

This entity is used to describe the actions of a particular user with a particular entity. These actions are ones that make a change to the database, such as adding a tuple, deleting a tuple, or updating a tuple. The primary key is the “transaction_code”. These entities will be automatically generated by the database.

Attributes:

- transaction_code: This is a unique NUMERIC code allocated to the database transaction. It is the primary key and can’t be null. The database will generate this code, or it can be entered manually if there is an error.
- entity: This is the VARCHAR name of the entity that was changed. It can’t be null.
- entity_code: This is the NUMERIC code of the entity that was changed, be it the user_code, expense_code, etc. This can’t be null.
- transacted_by: This is the code of the database user that made the change to the database. It also can’t be null.
- transaction_date_time: This is the timestamp of the database transaction. This can’t be null. It is in a TIMESTAMP data type.
- transaction_description: This is a VARCHAR description of the database transaction. This can be left null.
- transaction_notes: These are notes regarding the transaction. It is a VARCHAR data type. This can be left null if preferred.

Relationship and Cardinality Description

Relationship: “list” between PROPERTY and RENTAL_CONTRACT

- Cardinality: 1:M between PROPERTY (mandatory) and RENTAL_CONTRACT (optional)
- Business Rules:
    - A PROPERTY (mandatory) may be listed on zero, one, or many RENTAL_CONTRACTs-past and present (optional).
    - A RENTAL_CONTRACT (optional) must list one and only one PROPERTY (mandatory).

Relationship: “incur” between PROPERTY and EXPENSE

- Cardinality: 1:M between PROPERTY (mandatory) and EXPENSE (optional)
- Business rules:
    - A PROPERTY (mandatory) may incur zero, one or many EXPENSEs (optional).
    - An EXPENSE (optional) must be incurred by one and only one PROPERTY (mandatory).

Relationship: “receive” between PROPERTY and INCOME

- Cardinality: 1:M between PROPERTY (mandatory) and INCOME (optional)
- Business rules:
    - A PROPERTY (mandatory) may receive zero, one or many INCOMEs (optional).
    - An INCOME (optional) must be received by one and only one PROPERTY (mandatory).

Relationship: “sign” between PRIMARY_TENANT and RENTAL_CONTRACT

- Cardinality: 1:M between PRIMARY_TENANT (mandatory) and RENTAL_CONTRACT (optional)
- Business rules:
    - A PRIMARY_TENANT (mandatory) may sign zero, one or many RENTAL_CONTRACTs (optional) -past and present.
    - A RENTAL_CONTRACT (optional) must be signed by one and only one PRIMARY_TENANT (mandatory).

Relationship:" conduct" between USER_LIST and DATABASE_TRANSACTIONs.

- Cardinality: 1:M between USER_LIST (mandatory) and DATABASE_TRANSACTIONs (optional)
- Business rules:
    - A USER_LIST entity (mandatory) may conduct zero, one, or many DATABASE_TRANSACTIONs (optional).
    - A DATABASE_TRANSACTION (optional) is conducted by one and only one USER_LIST entity (mandatory).

Relationship: "manage" between PROPERTY and DATABASE_TRANSACTIONs.

- Cardinality: 1:M between PROPERTY (optional) and DATABASE_TRANSACTIONs (mandatory)
- Business rules:
    - A DATABASE_TRANSACTION (mandatory) may manage one and only one PROPERTY (optional).
    - A PROPERTY (optional) may be managed by one or many DATABASE_TRANSACTIONs (mandatory).

Relationship: "manage" between PRIMARY_TENANT and DATABASE_TRANSACTION.

- Cardinality: 1:M between PRIMARY_TENANT (optional) and DATABASE_TRANSACTION (mandatory)
- Business rules:
    - A DATABASE_TRANSACTION (mandatory) may manage zero or one PRIMARY_TENANT (optional).
    - A PRIMARY_TENANT (optional) may be managed by one or many DATABASE_TRANSACTIONs (mandatory).

Relationship: “manage” between EXPENSE and DATABASE_TRANSACTION.

- Cardinality: 1:M between EXPENSE (optional) and DATABASE_TRANSACTION (mandatory)
- Business rules:
    - A DATABASE_TRANSACTION (mandatory) may manage zero or one EXPENSE (optional).
    - An EXPENSE (optional) must be managed by one or many DATABASE_TRANSACTIONs (mandatory).

Relationship: “manage” between INCOME and DATABASE_TRANSACTION.

- Cardinality: 1:M between INCOME (optional) and DATABASE_TRANSACTION (mandatory)
- Business rules:
    - A DATABASE_TRANSACTION (mandatory) may manage zero or one INCOME (optional).
    - An INCOME (optional) must be managed by one or many DATABASE_TRANSACTIONs (mandatory).

Relationship: “manage” between RENTAL_CONTRACT and DATABASE_TRANSACTION.

- Cardinality: 1:M between RENTAL_CONTRACT (optional) and DATABASE_TRANSACTION (mandatory)
- Business rules:
    - A DATABASE_TRANSACTION (mandatory) may manage zero or one RENTAL_CONTRACT (optional).
    - A RENTAL_CONTRACT (optional) must be managed by one or many DATABASE_TRANSACTIONs (mandatory).

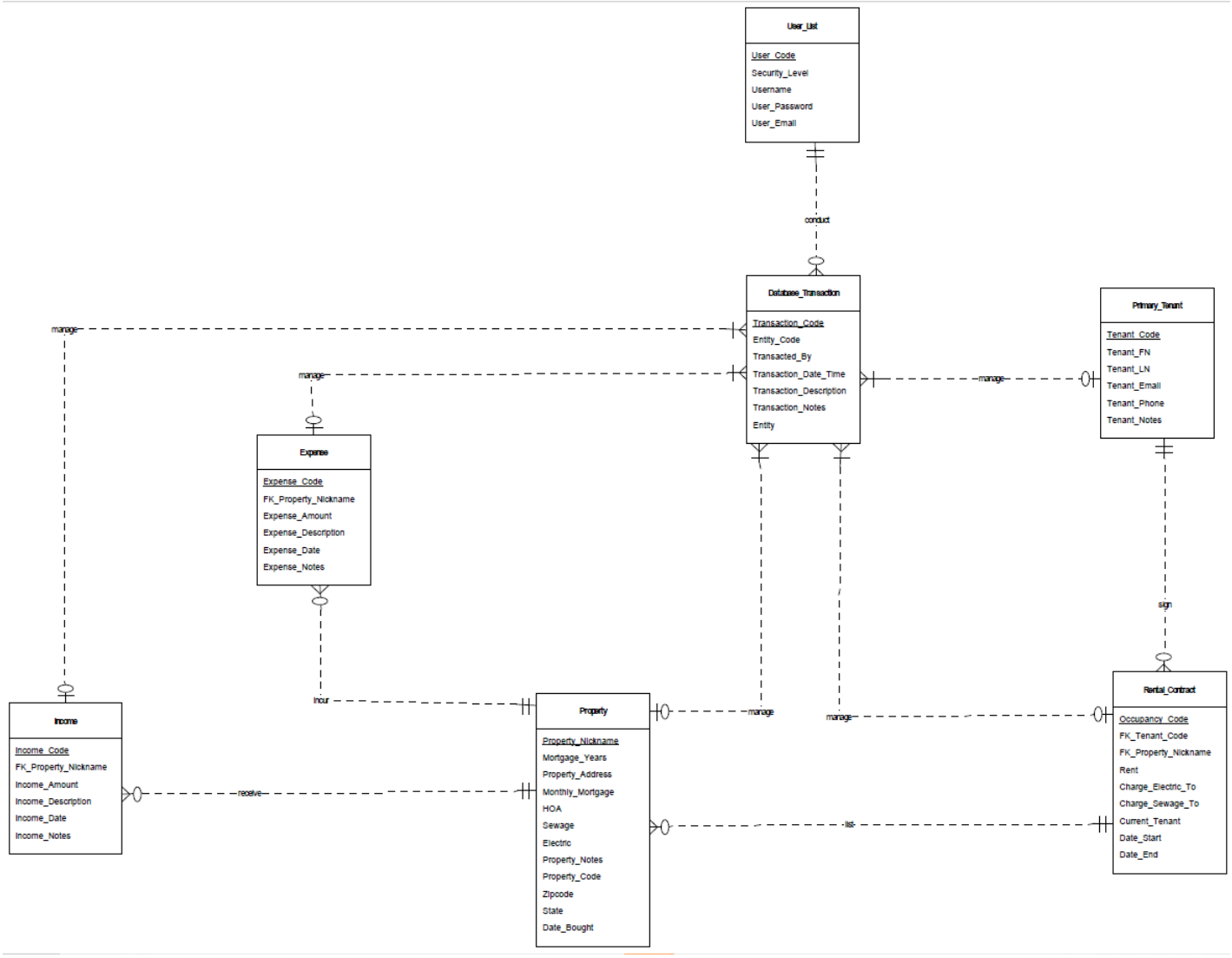
Assumptions and Special Considerations

In my project, I have 7 entities-one more than the maximum listed on the project outline. The reason for this extra entity, DATABASE_TRANSACTION, is to relate the USER_LIST entity with the other entities-resolving many-to-many (M:N) relationships between USER_LIST and other entities.

As the entity RENTAL_CONTRACT has an attribute for the current_tenant, an additional business rule not reflected in the ERD is that there can only be one RENTAL_CONTRACT with the current_tenant set to “Y” per PROPERTY entity. However, as a PRIMARY_TENANT may be renting more than one PROPERTY, a PRIMARY_TENANT can be listed as the tenant (via tenant_code) in more than one current (current_tenant = ‘Y’) RENTAL_CONTRACT, however the property_nickname (referring to PROPERTY entity’s Property_Nickname attribute) on the RENTAL_CONTRACTs will be different. This would be an unusual situation, but it is possible.

8. Detailed Database design

a. Entity Relationship Diagram (attached at Appendix A for a larger image)



b. DDL Source Code

--DDL and DML script by RENE GLIDDON for DBST 651, completed on 11/01/2022

--DDL Script

SET ECHO ON;

```
SET SERVEROUTPUT ON;

SET LINESIZE 150;

SET PAGESIZE 200;


--Drops delete objects so you can run the same script multiple times


--Drop Tables

DROP TABLE user_list CASCADE CONSTRAINTS;

DROP TABLE database_transaction CASCADE CONSTRAINTS;

DROP TABLE primary_tenant CASCADE CONSTRAINTS;

DROP TABLE property CASCADE CONSTRAINTS;

DROP TABLE rental_contract CASCADE CONSTRAINTS;

DROP TABLE expense CASCADE CONSTRAINTS;

DROP TABLE income CASCADE CONSTRAINTS;


--Drop sequences

DROP SEQUENCE user_code_Seq;

DROP SEQUENCE transaction_code_Seq;

DROP SEQUENCE tenant_code_Seq;

DROP SEQUENCE expense_code_Seq;

DROP SEQUENCE income_code_Seq;

DROP SEQUENCE occupancy_code_Seq;

DROP SEQUENCE property_code_Seq;
```

```
--Create the tables


--Create user_list table
CREATE TABLE user_list
(
  user_code        NUMERIC NOT NULL,
  security_level    VARCHAR (10) NOT NULL,
  username         VARCHAR (30),
  user_password     VARCHAR (10),
  user_email       VARCHAR(30),
  --This will ensure a UNIQUE primary key
  PRIMARY KEY (user_code)
);


--Create database_transaction table
CREATE TABLE database_transaction
(
  transaction_code          NUMERIC NOT NULL,
  entity                   VARCHAR (20) NOT NULL,
  entity_code              NUMERIC NOT NULL,
  transacted_by            VARCHAR(20) NOT NULL,
  transaction_date_time      TIMESTAMP(2) NOT NULL,
  transaction_description    VARCHAR(30),
  transaction_notes         VARCHAR(30),
```

```
  --This will ensure a UNIQUE primary key
  PRIMARY KEY (transaction_code)
);

--Create primary_tenant table
CREATE TABLE primary_tenant
(
  tenant_code              NUMERIC NOT NULL,
  tenant_fn                VARCHAR (10) NOT NULL,
  tenant_ln                VARCHAR(10) NOT NULL,
  tenant_email             VARCHAR(30),
  tenant_phone             VARCHAR(20) NOT NULL,
  tenant_notes             VARCHAR(100),
  --This will ensire a UNIQUE primary key
  PRIMARY KEY (tenant_code)
);

--Create property table
CREATE TABLE property
(
  property_code            NUMERIC NOT NULL,
  property_nickname        VARCHAR(50) NOT NULL,
  mortgage_years           DECIMAL (8,2) NOT NULL,
  property_address         VARCHAR(50) NOT NULL,
```

```
 state                VARCHAR(2),
 zipcode              VARCHAR (5),
 monthly_mortgage     DECIMAL(8,2) NOT NULL,
 hoa                  DECIMAL(8,2) NOT NULL,
 date_bought          DATE,
 sewage               VARCHAR(30),
 electric             VARCHAR(30),
 property_notes       VARCHAR(100),
 --This will ensure a UNIQUE primary key
 PRIMARY KEY (property_nickname)
);



--Create rental_contract table
CREATE TABLE rental_contract
(
 occupancy_code       NUMERIC NOT NULL,
 fk_tenant_code       NUMERIC NOT NULL,
 fk_property_nickname VARCHAR(50) NOT NULL,
 rent                 DECIMAL(8,2) NOT NULL,
 charge_electric_to   VARCHAR(20),
 charge_sewage_to     VARCHAR(30),
 current_tenant       CHAR(1),
 date_start           DATE NOT NULL,
```

```
 date_end                DATE NOT NULL,
 --This will ensure a UNIQUE primary key
 PRIMARY KEY (occupancy_code),
 FOREIGN KEY (fk_tenant_code) REFERENCES primary_tenant(tenant_code)
);

--Create expense table
CREATE TABLE expense
(
 expense_code            NUMERIC NOT NULL,
 fk_property_nickname    VARCHAR(50) NOT NULL,
 expense_amount          DECIMAL(8,2) NOT NULL,
 expense_description     VARCHAR(50) NOT NULL,
 expense_date            DATE NOT NULL,
 expense_notes           VARCHAR(200),
 --This will ensure a UNIQUE primary key
 PRIMARY KEY (expense_code),
 FOREIGN KEY (fk_property_nickname) REFERENCES property(property_nickname)
);

--Create income table
CREATE TABLE income
(
 income_code             NUMERIC NOT NULL,
```

```
 fk_property_nickname      VARCHAR(50) NOT NULL,
 income_amount             DECIMAL(8,2) NOT NULL,
 income_description        VARCHAR(50) NOT NULL,
 income_date               DATE NOT NULL,
 income_notes              VARCHAR(100),
 --This will ensure a UNIQUE primary key
 PRIMARY KEY (income_code),
 FOREIGN KEY (fk_property_nickname) REFERENCES property(property_nickname)
);

--Create sequences

--Create property_code sequence
CREATE SEQUENCE  property_code_Seq
START WITH 1
INCREMENT BY 1;

--Create user sequence
CREATE SEQUENCE user_code_Seq
START WITH 100
INCREMENT BY 1;

--Create database_transcation sequence
CREATE SEQUENCE transaction_code_Seq
```

```
START WITH 1

INCREMENT BY 1;


--Create tenant_code sequence

CREATE SEQUENCE tenant_code_Seq

START WITH 1

INCREMENT BY 1;


--Create occupancy_code sequence

CREATE SEQUENCE  occupancy_code_Seq

START WITH 100

INCREMENT BY 1;


--Create expense_code sequence

CREATE SEQUENCE  expense_code_Seq

START WITH 1

INCREMENT BY 1;


--Create income_code sequence

CREATE SEQUENCE  income_code_Seq

START WITH 1

INCREMENT BY 1;


--Indexes
```

```
--Create indexes for foreign keys

--expense table
CREATE INDEX expense_property_nickname_index
ON expense(fk_property_nickname);

--income table
CREATE INDEX income_property_nickname_index
ON income(fk_property_nickname);

--rental_contract table
CREATE INDEX rental_tenant_index
ON rental_contract(fk_tenant_code);

CREATE INDEX rental_property_index
ON rental_contract(fk_property_nickname);

--Add some audit columns to the tables

--user_list
ALTER TABLE user_list
ADD (
created_by VARCHAR2(30),
```

```
date_created DATE,
modified_by VARCHAR2(30),
date_modified DATE
);

--property
ALTER TABLE property
ADD (
created_by VARCHAR2(30),
date_created DATE,
modified_by VARCHAR2(30),
date_modified DATE
);

--primary_tenant
ALTER TABLE primary_tenant
ADD (
created_by VARCHAR2(30),
date_created DATE,
modified_by VARCHAR2(30),
date_modified DATE
);

--rental_contract
```

```
ALTER TABLE rental_contract
ADD (
created_by VARCHAR2(30),
date_created DATE,
modified_by VARCHAR2(30),
date_modified DATE
);

--expense
ALTER TABLE expense
ADD (
created_by VARCHAR2(30),
date_created DATE,
modified_by VARCHAR2(30),
date_modified DATE
);

--income
ALTER TABLE income
ADD (
created_by VARCHAR2(30),
date_created DATE,
modified_by VARCHAR2(30),
date_modified DATE
```

```
);

--database_transaction table will not have audit columns as it is an audit table

--Views

--View each table (except database_transaction) without audit columns

--user_list: this view shows basic user_list information without audit columns
CREATE OR REPLACE VIEW view_user_list AS
SELECT user_code, security_level, username, user_password, user_email FROM user_list;

--property: this view shows basic property information without audit columns
CREATE OR REPLACE VIEW view_property AS
SELECT property_code, property_nickname, mortgage_years, property_address,
monthly_mortgage, hoa, sewage, electric, property_notes FROM property;

--primary_tenant: this view shows basic primary_tenant information without audit columns
CREATE OR REPLACE VIEW view_primary_tenant AS
SELECT tenant_code, tenant_fn, tenant_ln, tenant_email, tenant_phone, tenant_notes FROM
primary_tenant;

--rental_contract: this view shows basic rental_contract information without audit columns
CREATE OR REPLACE VIEW view_rental_contract AS
```

```
SELECT occupancy_code, fk_tenant_code, fk_property_nickname, rent, charge_electric_to,
charge_sewage_to, current_tenant, date_start, date_end FROM rental_contract;

--expense: this view shows basic expense information without audit columns
CREATE OR REPLACE VIEW view_expense AS
SELECT expense_code, fk_property_nickname, expense_amount, expense_description,
expense_date, expense_notes FROM expense;

--income: this view shows basic income information without audit columns
CREATE OR REPLACE VIEW view_income AS
SELECT income_code, fk_property_nickname, income_amount, income_description, income_date,
income_notes FROM income;

--Views that manipulate data to show helpful information to the property owner

--Tenant phone view. This view is to create a phone list for the property owner to view and call if
necessary
CREATE OR REPLACE VIEW tenant_phone AS
SELECT  tenant_fn || ' ' ||  tenant_ln AS "Name", tenant_phone AS "Phone Number" FROM
primary_tenant;

--Expenses by tenant occupancy view. The purpose of this view is to view all expenses incurred by
tenant. This is to see if a tenant is causing an abnormal amount of expenses due to damages
CREATE OR REPLACE VIEW tenant_expense AS
```

```
SELECT c.tenant_fn || ' ' || c.tenant_ln AS "Name", a.fk_tenant_code AS "tenant_code",
b.expense_description, b.expense_amount
FROM primary_tenant c
INNER JOIN rental_contract a ON c.tenant_code = a.fk_tenant_code
INNER JOIN expense b ON a.fk_property_nickname = b.fk_property_nickname;

--Income paid by each tenant view.
--First set the column size so that it displays in a readable fashion
column c1 heading "Name" format a15
column c2 heading "Property" format a15
column c3 heading "Paid On" format a15
column c4 heading "Description" format a15
column c5 heading "Amount" format a15
column c6 heading "Contract Dates" format a20

--Create the view of income per tenant per property to see how much income has been generated
by each tenant per property
--This allows the property owner to see if rent payments have been missed, and if so, if late fees
were collected
CREATE OR REPLACE VIEW income_per_tenant_per_rental_contract AS
SELECT a.tenant_fn || ' ' || a.tenant_ln c1, b.fk_property_nickname c2, c.income_date c3,
c.income_description c4, c.income_amount c5, b.date_start || ' ' || b.date_end c6
FROM primary_tenant a
INNER JOIN rental_contract b ON a.tenant_code = b.fk_tenant_code
```

```
INNER JOIN income c ON b.fk_property_nickname = c.fk_property_nickname

WHERE c.income_date <= (SELECT CURRENT_DATE FROM dual) AND c.income_date >= b.date_start;


--triggers


--audit table database_transaction and sequence triggers for each table


--property table


--trigger to populate the database_transaction table when a row is inserted into property, as well as

to create a sequenced primary key when it is null

CREATE OR REPLACE TRIGGER property_insert_trig

BEFORE INSERT

ON property

FOR EACH ROW

DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;

BEGIN

IF :NEW.property_code IS NULL THEN

:NEW.property_code := property_code_Seq.NEXTVAL;

END IF;

entity := :NEW.property_code;

SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;

SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
```

```
INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
VALUES (transaction_code_Seq.nextVal, 'property', entity, username, date_time, 'Property added');
END;
/

--trigger to populate the database_transaction table when a row is updated in property
CREATE OR REPLACE TRIGGER property_update_trig
BEFORE UPDATE ON property
FOR EACH ROW
DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
BEGIN
IF :NEW.property_code IS NULL THEN
:NEW.property_code := property_code_Seq.NEXTVAL;
END IF;
entity := :NEW.property_nickname;
SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
VALUES (transaction_code_Seq.nextVal, 'property', entity, username, date_time, 'Property
updated');
END;
/
```

```
--primary_tenant table


--trigger to populate the database_transaction table when a row is inserted into primary_tenant as
well as to create a sequenced primary key when it is null
CREATE OR REPLACE TRIGGER primary_tenant_insert_trig
BEFORE INSERT
ON primary_tenant
FOR EACH ROW
DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
BEGIN
IF :NEW.tenant_code IS NULL THEN
:NEW.tenant_code := tenant_code_Seq.NEXTVAL;
END IF;
entity := :NEW.tenant_code;
SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
VALUES (transaction_code_Seq.nextVal, 'primary_tenant', entity, username, date_time, 'Primary
tenant added');
END;
/
```

```
--trigger to populate the database_transaction table when a row is updated in primary_tenant table
CREATE OR REPLACE TRIGGER primary_tenant_update_trig
AFTER UPDATE ON primary_tenant
FOR EACH ROW
DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
BEGIN
entity := :NEW.tenant_code;
SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
VALUES (transaction_code_Seq.nextVal, 'primary tenant', entity, username, date_time, 'Property
updated');
END;
/


--rental_contract table


--trigger to populate the database_transaction table when a row is inserted into rental_contract as
well as to create a sequenced primary key when it is null
CREATE OR REPLACE TRIGGER rental_contract_insert_trig
BEFORE INSERT
ON rental_contract
FOR EACH ROW
```

```
DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
BEGIN
IF :NEW.occupancy_code IS NULL THEN
:NEW.occupancy_code := occupancy_code_Seq.NEXTVAL;
END IF;
entity := :NEW.occupancy_code;
SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
VALUES (transaction_code_Seq.nextVal, 'Rental contract', entity, username, date_time, 'Rental
contract added');
END;
/

--trigger to populate the database_transaction table when a row is updated in rental_contract table
CREATE OR REPLACE TRIGGER rental_contract_update_trig
AFTER UPDATE ON rental_contract
FOR EACH ROW
DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
BEGIN
entity := :NEW.occupancy_code;
SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
```

```
INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
VALUES (transaction_code_Seq.nextVal, 'Rental contract', entity, username, date_time, 'Rental
contract updated');
END;
/


--expense table


--trigger to populate the database_transaction table when a row is inserted into the expense table
as well as to create a sequenced primary key when it is null
CREATE OR REPLACE TRIGGER expense_insert_trig
BEFORE INSERT
ON expense
FOR EACH ROW
DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
BEGIN
IF :NEW.expense_code IS NULL THEN
:NEW.expense_code := expense_code_Seq.NEXTVAL;
END IF;
entity := :NEW.expense_code;
SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
```

```
INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
VALUES (transaction_code_Seq.nextVal, 'Expense', entity, username, date_time, 'Expense added');
END;
/

--trigger to populate the database_transaction table when a row is updated in expense table
CREATE OR REPLACE TRIGGER expense_update_trig
AFTER UPDATE ON expense
FOR EACH ROW
DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
BEGIN
entity := :NEW.expense_code;
SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
VALUES (transaction_code_Seq.nextVal, 'expense', entity, username, date_time, 'expense updated');
END;
/

--income table
```

```
--trigger to populate the database_transaction table when a row is inserted into rental_contract as
well as to create a sequenced primary key when it is null
CREATE OR REPLACE TRIGGER income_insert_trig
BEFORE INSERT
ON income
FOR EACH ROW
DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
BEGIN
IF :NEW.income_code IS NULL THEN
:NEW.income_code := income_code_Seq.NEXTVAL;
END IF;
entity := :NEW.income_code;
SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
VALUES (transaction_code_Seq.nextVal, 'Income', entity, username, date_time, 'Income added');
END;
/



--trigger to populate the database_transaction table when a row is updated in income table
CREATE OR REPLACE TRIGGER income_update_trig
AFTER UPDATE ON income
```

```
FOR EACH ROW
DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
BEGIN
entity := :NEW.income_code;
SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
VALUES (transaction_code_Seq.nextVal, 'Income', entity, username, date_time, 'Income updated');
END;
/


--user_list table


--trigger to populate the database_transaction table when a row is inserted into user_list as well as
to create a sequenced primary key when it is null
CREATE OR REPLACE TRIGGER user_list_insert_trig
BEFORE INSERT
ON user_list
FOR EACH ROW
DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
BEGIN
IF :NEW.user_code IS NULL THEN
:NEW.user_code := user_code_Seq.NEXTVAL;
```

```
END IF;
entity := :NEW.user_code;
SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
VALUES (transaction_code_Seq.nextVal, 'User', entity, username, date_time, 'User added');
END;
/

--trigger to populate the database_transaction table when a row is updated in rental_contract table
CREATE OR REPLACE TRIGGER user_list_update_trig
AFTER UPDATE ON user_list
FOR EACH ROW
DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
BEGIN
entity := :NEW.user_code;
SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
VALUES (transaction_code_Seq.nextVal, 'User', entity, username, date_time, 'User updated');
END;
/
```

c. DML and Query Source Code

```
--DML Script


--insert some sample data


--parent tables


--primary_tenant table
INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email, tenant_phone,
tenant_notes)
VALUES (tenant_code_Seq.nextVal, 'Lisa', 'Smith', 'lisasmith@gmail.com', '222 112 9090', 'requested
to pay on the 5th of each month instead of the 1st');

INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email, tenant_phone,
tenant_notes)
VALUES (tenant_code_Seq.nextVal, 'Amy', 'Adams', 'amyadams@gmail.com', '222 111 1010', 'has a
pet ferret');

INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email, tenant_phone,
tenant_notes)
VALUES (tenant_code_Seq.nextVal, 'Barbara', 'Bayleaf', 'barbay@gmail.com', '123 102 3090',
'requested the kitchen be repainted');
```

```
INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email, tenant_phone,
tenant_notes)
VALUES (tenant_code_Seq.nextVal, 'Curtis', 'Caveman', 'curtcave@gmail.com', '200 312 8766',
'drummer. may new to add soundproofing if neighbors complain');

INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email, tenant_phone,
tenant_notes)
VALUES (tenant_code_Seq.nextVal, 'Darlene', 'Davis', 'davisthedarling@gmail.com', '232 122 9990',
'single mom');

INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email, tenant_phone,
tenant_notes)
VALUES (tenant_code_Seq.nextVal, 'Elliot', 'Earl', 'earlelli@gmail.com', '213 100 0090', 'neighbors
complain he is weird');

INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email, tenant_phone,
tenant_notes)
VALUES (tenant_code_Seq.nextVal, 'Farrah', 'Fawcett', 'farfaw@gmail.com', '122 000 9090', '2
dogs');

INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email, tenant_phone,
tenant_notes)
VALUES (tenant_code_Seq.nextVal, 'Gary', 'Indiana', 'onedollarhouse@gmail.com', '192 222 1190',
'inherited from previous owner');
```

```
INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email, tenant_phone,
tenant_notes)
VALUES (tenant_code_Seq.nextVal, 'Harry', 'Smith', 'harhar@gmail.com', '200 162 3489', 'reliable
tenant');

INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email, tenant_phone,
tenant_notes)
VALUES (tenant_code_Seq.nextVal, 'Ingrid', 'Indigo', 'indigo@gmail.com', '415 199 9090', 'repainted
the livingroom purple. take out of deposit when she leaves');

--user_list table
INSERT INTO user_list (user_code, security_level)
VALUES (user_code_Seq.nextVal, 'tenant');

INSERT INTO user_list (user_code, security_level)
VALUES (user_code_Seq.nextVal, 'owner');

INSERT INTO user_list (user_code, security_level)
VALUES (user_code_Seq.nextVal, 'tenant');

INSERT INTO user_list (user_code, security_level)
VALUES (user_code_Seq.nextVal, 'owner');
```

```
INSERT INTO user_list (user_code, security_level)
VALUES (user_code_Seq.nextVal, 'tenant');

INSERT INTO user_list (user_code, security_level)
VALUES (user_code_Seq.nextVal, 'tenant');

INSERT INTO user_list (user_code, security_level)
VALUES (user_code_Seq.nextVal, 'tenant');

INSERT INTO user_list (user_code, security_level)
VALUES (user_code_Seq.nextVal, 'tenant');

INSERT INTO user_list (user_code, security_level)
VALUES (user_code_Seq.nextVal, 'tenant');

INSERT INTO user_list (user_code, security_level)
VALUES (user_code_Seq.nextVal, 'tenant');

--property table

INSERT INTO property (property_code, property_nickname, mortgage_years, property_address,
state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
VALUES (property_code_Seq.nextVal, 'Alaska', 30, '123 Alaska Street, Tacoma', 'WA', '98499',
900.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'Needs renovations');
```

```
INSERT INTO property (property_code, property_nickname, mortgage_years, property_address,
state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
VALUES (property_code_Seq.nextVal, 'Edgewood Unit 1', 30, '123 Edgewood Street, Edgewood',
'MD', '21040', 700.00, 0.00, 'YuckYuck Sewage', 'Tacoma Electric', 'mortgage split with unit 2');

INSERT INTO property (property_code, property_nickname, mortgage_years, property_address,
state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
VALUES (property_code_Seq.nextVal, 'Edgewood Unit 2', 30, '123 Edgewood Street, Edgewood',
'MD',  '21040', 700.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split with unit 1');

INSERT INTO property (property_code, property_nickname, mortgage_years, property_address,
state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
VALUES (property_code_Seq.nextVal, 'Yakima Unit 1', 30, '123 Yakima Street, Overland Park', 'KS',
'66204', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways');

INSERT INTO property (property_code, property_nickname, mortgage_years, property_address,
state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
VALUES (property_code_Seq.nextVal, 'Yakima Unit 2', 30, '123 Yakima Street, Overland Park', 'KS',
'66204', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways');

INSERT INTO property (property_code, property_nickname, mortgage_years, property_address,
state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
```

VALUES (property_code_Seq.nextVal, 'Yakima Unit 3', 30, '123 Yakima Street, Overland Park', 'KS', '66204', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways');

INSERT INTO property (property_code, property_nickname, mortgage_years, property_address, state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
VALUES (property_code_Seq.nextVal, 'Yakima Unit 4', 30, '123 Yakima Street, Overland Park', 'KS', '66204', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways');

INSERT INTO property (property_code, property_nickname, mortgage_years, property_address, state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
VALUES (property_code_Seq.nextVal, 'Monroe Unit 1', 30, '123 Monroe Street, Bel Air', 'MD', '21014', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways. high-end renovation in 2021');

INSERT INTO property (property_code, property_nickname, mortgage_years, property_address, state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
VALUES (property_code_Seq.nextVal, 'Monroe Unit 2', 30, '123 Monroe Street, Bel Air', 'MD', '21014', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways');

INSERT INTO property (property_code, property_nickname, mortgage_years, property_address, state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
VALUES (property_code_Seq.nextVal, 'Monroe Unit 3', 30, '123 Monroe Street, Bel Air', 'MD', '21014', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways. Recently renovated in 2022');

```
INSERT INTO property (property_code, property_nickname, mortgage_years, property_address,
state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
VALUES (property_code_Seq.nextVal, 'Monroe Unit 4', 30, '123 Monroe Street, Bel Air', 'MD',
'21014', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways. undergoing
renovation');

--child tables

--rental_contract table
INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
VALUES (occupancy_code_seq.nextval, 1, 'Alaska', 1100.00, 'Tenant', 'Tenant', 'Y', DATE '2020-01-
01', DATE '2020-12-31');

INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
VALUES (occupancy_code_seq.nextval, 2, 'Edgewood Unit 1', 800.00, 'Tenant', 'Tenant', 'N', DATE
'2020-01-01', DATE '2020-12-31');

INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
VALUES (occupancy_code_seq.nextval, 2, 'Edgewood Unit 1', 900.00, 'Tenant', 'Tenant', 'N', DATE
'2021-01-01', DATE '2021-12-31');
```

```
INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
VALUES (occupancy_code_seq.nextval, 2, 'Edgewood Unit 1', 1100.00, 'Tenant', 'Tenant', 'Y', DATE
'2022-01-01', DATE '2022-12-31');

INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
VALUES (occupancy_code_seq.nextval, 3, 'Edgewood Unit 2', 950.00, 'Tenant', 'Tenant', 'Y', DATE
'2022-01-01', DATE '2022-12-31');

INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
VALUES (occupancy_code_seq.nextval, 4, 'Yakima Unit 1', 1100.00, 'Tenant', 'Tenant', 'Y', DATE
'2022-01-01', DATE '2022-12-31');

INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
VALUES (occupancy_code_seq.nextval, 5, 'Yakima Unit 2', 550.00, 'Tenant', 'Tenant', 'Y', DATE '2022-
01-01', DATE '2022-12-31');

INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
```

```
VALUES (occupancy_code_seq.nextval, 6, 'Yakima Unit 3', 1800.00, 'Tenant', 'Tenant', 'Y', DATE
'2022-01-01', DATE '2022-12-31');

INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
VALUES (occupancy_code_seq.nextval, 7, 'Yakima Unit 4', 1150.00, 'Tenant', 'Tenant', 'Y', DATE
'2022-01-01', DATE '2022-12-31');

INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
VALUES (occupancy_code_seq.nextval, 8, 'Monroe Unit 1', 12800.00, 'Tenant', 'Tenant', 'Y', DATE
'2022-01-01', DATE '2022-12-31');

INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
VALUES (occupancy_code_seq.nextval, 9, 'Monroe Unit 2', 800.00, 'Tenant', 'Tenant', 'Y', DATE
'2020-01-01', DATE '2020-12-31');

INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
VALUES (occupancy_code_seq.nextval, 10, 'Monroe Unit 3', 900.00, 'Tenant', 'Tenant', 'Y', DATE
'2020-01-01', DATE '2020-12-31');

--expense table
```

```
INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
VALUES (expense_code_Seq.nextval, 'Monroe Unit 2', 150.00, 'Blocked Drain', DATE '2020-03-12',
'Hair in shower drain');

INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
VALUES (expense_code_Seq.nextval, 'Alaska', 100.00, 'Door handles', DATE '2020-03-12', 'tenant is
afraid of doorknobs');

INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
VALUES (expense_code_Seq.nextval, 'Edgewood Unit 1', 2450.00, 'Exterminator', DATE '2022-05-12',
'infestation due to secret litter of ferret pups in the drywall');

INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
VALUES (expense_code_Seq.nextval, 'Edgewood Unit 2', 1150.00, 'Paint and Painter', DATE '2022-
09-12', 'Kitchen repainted due to old cooking stains on wall');

INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
```

```
VALUES (expense_code_Seq.nextval, 'Edgewood Unit 2', 2150.00, 'Exterminator', DATE '2022-08-01',
'Ferrets found in wood panelling. Flea infestation due to this. Ferrets taken to Unit 1 to their
mother');

INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
VALUES (expense_code_Seq.nextval, 'Edgewood Unit 2', 1550.00, 'Wall repair', DATE '2020-07-27',
'Hole between Unit 1 and Unit 2 discovered');

INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
VALUES (expense_code_Seq.nextval, 'Monroe Unit 1', 350.00, 'Chocolate fountain repair', DATE
'2022-03-12', 'Chocolate fountain in bathroom needed a new pipe');

INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
VALUES (expense_code_Seq.nextval, 'Monroe Unit 1', 250.00, 'Chandeliar Reshining', DATE '2022-
05-12', 'Chandelier needed reshining/repolishing');

INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
VALUES (expense_code_Seq.nextval, 'Monroe Unit 1', 100.00, 'Blocked Drain', DATE '2022-03-12',
'Diamonds from toilet seat dislodged into drain. Needed retrival');
```

```
INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
VALUES (expense_code_Seq.nextval, 'Monroe Unit 1', 550.00, 'Tank cleaning', DATE '2022-05-12',
'Jellyfish tank in the lobby required annual cleaning');

INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
VALUES (expense_code_Seq.nextval, 'Alaska', 850.00, 'New carpets', DATE '2022-02-02', 'new
carpets installed');

INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
VALUES (expense_code_Seq.nextval, 'Yakima Unit 1', 1850.00, 'Porch repair', DATE '2022-09-24',
'porch was rotting and needed replacement');

--income table
INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Alaska', 1100.00, 'Rent', DATE '2022-10-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Alaska', 1100.00, 'Rent', DATE '2022-09-01');
```

```
INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Alaska', 1100.00, 'Rent', DATE '2022-08-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Alaska', 1100.00, 'Rent', DATE '2022-07-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Alaska', 1100.00, 'Rent', DATE '2022-06-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Alaska', 1100.00, 'Rent', DATE '2022-05-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Alaska', 1100.00, 'Rent', DATE '2022-02-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Monroe Unit 1', 12800.00, 'Rent', DATE '2022-01-01');
```

```
INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Monroe Unit 1', 12800.00, 'Rent', DATE '2022-02-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Monroe Unit 1', 12800.00, 'Rent', DATE '2022-03-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Monroe Unit 1', 100.00, 'Late fee', DATE '2022-04-15');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Monroe Unit 1', 100.00, 'Late fee', DATE '2022-05-15');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Monroe Unit 1', 38400.00, 'Rents due to date', DATE '2022-06-
01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-01-01');
```

```
INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-02-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-03-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-04-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-05-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-06-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 100.00, 'Late Fee', DATE '2022-06-10');
```

```
INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-06-25');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-07-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-08-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-09-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-10-01');

INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Monroe Unit 3', 100.00, 'Late Fee', DATE '2022-01-10');
```

```
INSERT INTO income(income_code, fk_property_nickname, income_amount, income_description,
income_date)
VALUES (income_code_Seq.nextval, 'Yakima Unit 3', 100.00, 'Late Fee', DATE '2022-01-10');



--database_transaction not needed to be added as this is done automatically via trigger

--Database queries to show that objects were created

--table queries (but not other tables in the dbst connection)

--show the tables exist
SELECT table_name FROM user_tables WHERE sample_size IS NULL;

--Describe tables
--user_list table
DESCRIBE user_list;

--property table
DESCRIBE property;

--primary_tenant table
```

```
DESCRIBE primary_tenant;


--rental_contract table

DESCRIBE rental_contract;


--expense table

DESCRIBE expense;


--income table

DESCRIBE income;


--database_transaction table

DESCRIBE database_transaction;


--indexes

--show indexes exist

-- only objects created after 7 March are selected due to the STUDENT database objects showing up

as well

COLUMN a1 HEADING "Object Name" FORMAT a35

COLUMN a2 HEADING "Object Type" FORMAT a15

COLUMN a3 HEADING "Status" FORMAT a15

COLUMN a4 HEADING "Created" FORMAT a15


SELECT object_name a1, object_type a2,  status a3, created a4
```

```
FROM user_objects

WHERE created > '07-MAR-22'

AND GENERATED = 'N'

AND object_type = 'INDEX' ;


--triggers

--view triggers (selected only object_name to simplify viewing)

COLUMN b1 HEADING "Object Name" FORMAT a30

COLUMN b2 HEADING "Object Type" FORMAT a15

COLUMN b3 HEADING "Status" FORMAT a15

COLUMN b4 HEADING "Created" FORMAT a15


SELECT object_name b1, object_type b2,  status b3, created b4

FROM user_objects

WHERE object_type='TRIGGER';


--view views

--selected only object_name to simplify viewing

COLUMN d1 HEADING "Object Name" FORMAT a40

COLUMN d2 HEADING "Object Type" FORMAT a15

COLUMN d3 HEADING "Status" FORMAT a15

COLUMN d4 HEADING "Created" FORMAT a15


SELECT object_name d1, object_type d2,  status d3, created d4
```

```
FROM user_objects

WHERE object_type='VIEW';


--view indexes

--only object__name selected to simplify viewing, and only objects created after 7 March are

selected due to the STUDENT database objects showing up as well

COLUMN e1 HEADING "Object Name" FORMAT a20

COLUMN e2 HEADING "Object Type" FORMAT a15

COLUMN e3 HEADING "Status" FORMAT a15

COLUMN e4 HEADING "Created" FORMAT a15

SELECT object_name e1, object_type e2,  status e3, created e4 FROM user_objects WHERE

object_type='SEQUENCE' AND created > '07-MAR-22';



--Query 1: Select all columns and all rows from one table

SELECT *

FROM primary_tenant;


--Query 2: Select five columns and all rows from one table

SELECT fk_property_nickname, expense_amount, expense_description, expense_date,

expense_notes

FROM expense;


--Query 3: Select all columns from all rows from one view
```

```
SELECT *
FROM  income_per_tenant_per_rental_contract;

--Query 4: Using a join on 2 tables, select all columns and all rows from the tables without the use of
a Cartesian product
SELECT *
FROM rental_contract
INNER JOIN primary_tenant
ON rental_contract.fk_tenant_code = primary_tenant.tenant_code;

--Query 5: Select and order data retrieved from one table
--First set the column size so that it displays in a readable fashion
column c1 heading "Expense Code" format a10
column c2 heading "Property" format a20
column c3 heading "Expense Amount" format a10
column c4 heading "Description" format a20
column c5 heading "Date" format a10
column c6 heading "Notes" format a30

SELECT expense_code c1, fk_property_nickname c2, expense_amount c3, expense_description c4,
expense_date c5, expense_notes c6
FROM expense
ORDER BY expense_date ASC;
```

```
--Query 6: Using a join on 3 tables, select 5 columns from the 3 tables. Use syntax that would limit
the output to 10 rows
SELECT c.property_nickname AS "Property", b.tenant_fn AS "First name", b.tenant_ln AS "Last
name", a.rent AS "Rent", a.date_end AS "Contract End"
FROM rental_contract a
INNER JOIN primary_tenant b
ON a.fk_tenant_code = b.tenant_code
INNER JOIN property c
ON c.property_nickname = a.fk_property_nickname
WHERE b.tenant_code < 11;

--Query 7: Select distinct rows using joins on 3 tables
SELECT DISTINCT b.sewage, b.property_nickname AS "Properties", c.tenant_code AS "Tenant Code"
FROM rental_contract a
INNER JOIN property b
ON a.fk_property_nickname = b.property_nickname
INNER JOIN primary_tenant c
ON a.fk_tenant_code = c.tenant_code;

--Query 8: Use GROUP BY and HAVING in a select statement using one or more tables
SELECT b.sewage, COUNT(b.property_nickname) AS "Number of Properties", COUNT(c.tenant_code)
AS "Number of Contracts"
FROM rental_contract a
INNER JOIN property b
```

```
ON a.fk_property_nickname = b.property_nickname

INNER JOIN primary_tenant c

ON a.fk_tenant_code = c.tenant_code

GROUP BY b.sewage

HAVING COUNT(b.property_nickname) >3;


--Query 9: Use IN clause to select data from one or more tables

SELECT a.tenant_fn, a.tenant_ln, b.fk_tenant_code, c.property_nickname

FROM primary_tenant a

INNER JOIN rental_contract b

ON a.tenant_code = b.fk_tenant_code

INNER JOIN property c

ON b.fk_property_nickname = c.property_nickname

WHERE a.tenant_code

IN ('1' , '3', '9');


--Query 10: Select length of one column from one table

SELECT LENGTH (tenant_notes) AS "Length of Tenant Notes"

FROM primary_tenant;


--Query 11: Delete one record from one table. Use select statements to demonstrate the table

contents before and after the DELETE statement.

--Make sure you use ROLLBACK afterwards so that the data will not be physically removed
```

```
--Show the table before the DELETE statement

--First set the column size so that it displays in a readable fashion

column c1 heading "Expense Code" format a10

column c2 heading "Property" format a20

column c3 heading "Expense Amount" format a10

column c4 heading "Description" format a20

column c5 heading "Date" format a10

column c6 heading "Notes" format a30

SELECT expense_code c1, fk_property_nickname c2, expense_amount c3, expense_description c4,

expense_date c5, expense_notes c6 FROM expense;



--Delete one record

DELETE

FROM expense

WHERE expense_description = 'Wall repair';


--Show the table after the DELETE statement

--First set the column size so that it displays in a readable fashion

column c1 heading "Expense Code" format a10

column c2 heading "Property" format a20

column c3 heading "Expense Amount" format a10

column c4 heading "Description" format a20

column c5 heading "Date" format a10
```

```
column c6 heading "Notes" format a30
SELECT expense_code c1, fk_property_nickname c2, expense_amount c3, expense_description c4,
expense_date c5, expense_notes c6 FROM expense;


--Rollback after DELETE statement
--Hwoever, the rollback rolls back more than just the DELETE statement (my insert statements, etc.)
and causes issues when the script is run all together
--Hence, the rollback is commented out and must be uncommented, run individually (just for the
delete statement), and recommented to allow the entire scritp to be run without errors
--ROLLBACK;


-- Query 12: Update one record from one table. Use select statements to demonstrate the table
contents before and after the UPDATE statement.
--Make sure you use ROLLBACK afterwards so that the data will not be physically removed

--Show table contents before update
--First set the column size so that it displays in a readable fashion
column c1 heading "Expense Code" format a10
column c2 heading "Property" format a20
column c3 heading "Expense Amount" format a10
column c4 heading "Description" format a20
column c5 heading "Date" format a10
```

```
column c6 heading "Notes" format a30
SELECT expense_code c1, fk_property_nickname c2, expense_amount c3, expense_description c4,
expense_date c5, expense_notes c6 FROM expense;

--Update the table
UPDATE expense
SET expense_description = 'Ferret Whisperer'
WHERE expense_code = '5';

--Show table after update
--First set the column size so that it displays in a readable fashion
column c1 heading "Expense Code" format a10
column c2 heading "Property" format a20
column c3 heading "Expense Amount" format a10
column c4 heading "Description" format a20
column c5 heading "Date" format a10
column c6 heading "Notes" format a30
SELECT expense_code c1, fk_property_nickname c2, expense_amount c3, expense_description c4,
expense_date c5, expense_notes c6 FROM expense;

--Rollback the update
--(however, this undoes my table inserts as well, renderign the rest of this script problematic hence,
this rollback is commented out. )
```

```
--When using the update, it should be uncommented and rolled back individually (not the whole
script)
--ROLLBACK;

--Perform 8 Additional Advanced Queries

--Query 13: Calculate the minimum, maximum and average expense per zipcode
SELECT b.zipcode, MIN(a.expense_amount) AS "Min Expense", MAX(a.expense_amount) AS "Max
Expense", ROUND(AVG(a.expense_amount)) AS "Avg Expense"
FROM expense a
INNER JOIN property b
ON a.fk_property_nickname = b.property_nickname
GROUP BY b.zipcode;

--Query 14: List the current occupants

--Drop the view that will be created in this query
DROP VIEW current_occupancy;

--Create a view of the current occupants with valid contracts (valid in the current year) with the
property name and the dates of the contract
--This view is quite useful, and thus will be used in some queries that follow, to reduce redundant
work
CREATE VIEW current_occupancy AS
```

```
SELECT DISTINCT d.property_nickname, d.zipcode, b.tenant_code,  b.tenant_fn || ' ' || b.tenant_ln
AS "Tenant", c.date_start, c.date_end
FROM primary_tenant b
INNER JOIN rental_contract c
ON b.tenant_code = c.fk_tenant_code
INNER JOIN property d
ON d.property_nickname = c.fk_property_nickname
WHERE c.current_tenant = 'Y'
AND c.date_end > (
        SELECT add_months(
                (
                  SELECT sysdate
                  FROM dual
                 ), 12*-1
                ) FROM dual
        );


--Show the current occupants
SELECT *
FROM current_occupancy;


--Query 15: Show the average expense per zipcode in the last year-to-date
SELECT DISTINCT a.zipcode, ROUND(AVG(b.expense_amount)) AS "Avg Expense"
FROM property a
```

```
FULL JOIN expense b
ON a.property_nickname = b.fk_property_nickname
WHERE b.expense_date >= (SELECT TO_CHAR(ADD_MONTHS((SELECT sysdate FROM dual), 12*-
1),'dd-MON-yyyy') FROM dual)
AND b.expense_date <= (SELECT SYSDATE FROM DUAL)
GROUP BY a.zipcode
ORDER BY "Avg Expense" DESC;

--Query 16: Show tenants that are accumulating more than the average expenses in their zipcode in
the last year
--We will use the previously developed view "current_occupancy"

--Drop the view that will be created in this query
DROP VIEW expense_summary;

--first, calculate the expected versus actual expediture during the life of the tenant's contract (not
year to date, nor calendar year, but each contract year)
CREATE VIEW expense_summary AS
SELECT a.tenant_code, a.zipcode, NVL(ROUND(MONTHS_BETWEEN((SELECT sysdate FROM dual),
a.date_start)/12, 2) * b."Avg Expense", 0) AS "Expected", NVL(c."Expenses Accumulated",0) AS
"Actual"
FROM current_occupancy a
INNER JOIN average_per_zip b
ON a.zipcode = b.zipcode
```

```
FULL JOIN accumulated_expense c
ON c.tenant_code = a.tenant_code;

---Next, show the tenants that accumulated more expenses than what was expected/average using
the cumulative view "expense summary"  developed in this query
SELECT a.tenant_code, a.tenant_fn || ' ' || a.tenant_ln AS "Tenant", a.tenant_phone, b."Actual" -
b."Expected" AS "Over Average Per ZIP By"
FROM primary_tenant a
INNER JOIN expense_summary b
ON a.tenant_code = b.tenant_code
WHERE b."Actual" > b."Expected";

--Query 16: Calculate how much income per tenant has been accumulated from properties in
Washington state and Maryland
--We will use the previously developed view "current_occupancy" to assist

SELECT a.property_nickname, a."Tenant", a.tenant_code, SUM(b.income_amount) AS "Income
Accumulated"
FROM current_occupancy a
INNER JOIN income b
ON a.property_nickname = b.fk_property_nickname
INNER JOIN property c
ON a.property_nickname = c.property_nickname
WHERE c.state IN ('WA', 'MD')
```

```
GROUP BY a.property_nickname, a."Tenant", a.tenant_code;



--Query 17: List the tenants with current rental contracts who are behind in rent payments.
--We will use previously developed view, "current_occupancy", and develop new views based off of
this to further the queries
DROP VIEW months_of_rent;
DROP VIEW rent_due;
DROP VIEW due_and_received_rent;

--First, create a view to calculate the number of months of rent due for contracts that are current
CREATE VIEW months_of_rent AS
SELECT tenant_code, round (
                    MONTHS_BETWEEN(
                                (
                                  SELECT sysdate
                                  FROM dual
                                 ), date_start
                                )
                    ) AS "Rent Payments Due"
FROM current_occupancy;

--Calculate the dollar amount of rent due over the course of the rental contract to date per tenant
for current contracts, using the views created in this query
```

```
CREATE VIEW rent_due AS
SELECT a.tenant_code, a."Rent Payments Due" * b.rent AS "Rent Due"
FROM months_of_rent a
INNER JOIN rental_contract b
ON a.tenant_code = b.fk_tenant_code
WHERE b.date_end  > (
                SELECT sysdate FROM dual
               );


--Create a view showing the rent due and the rent received per current occupant, using the views
created in this query to this point
CREATE VIEW due_and_received_rent AS
SELECT a.tenant_code, c."Rent Due", SUM(b.income_amount) AS "Rent Received"
FROM current_occupancy a
RIGHT OUTER JOIN rent_due c
ON a.tenant_code = c.tenant_code
INNER JOIN rental_contract d
ON c.tenant_code = d.fk_tenant_code
INNER JOIN property e
ON d.fk_property_nickname = e.property_nickname
INNER JOIN income b
ON b.fk_property_nickname = e.property_nickname
WHERE b.income_date BETWEEN TO_DATE(a.date_start) AND TO_DATE(a.date_end)
AND b.income_description = 'Rent'
```

```
GROUP BY a.tenant_code,  c."Rent Due";


--Show the details of the tenants with current rental contracts who are behind in rent, using the
cumulative view created previously in this query
SELECT a.tenant_fn, a.tenant_ln, a.tenant_code, a.tenant_phone, a.tenant_email, ABS(b."Rent
Received" - b."Rent Due") AS "Overdue By"
FROM primary_tenant a
RIGHT OUTER JOIN due_and_received_rent b
ON a.tenant_code = b.tenant_code
WHERE b."Rent Due" > b."Rent Received";


/*Query 18: List tenants who need their rental contracts renewed.
This includes tenants do  not have current rental contracts, but are still paying rent or late fees, i.e.
are still actively living in the unit,
as well as tenants whose contracts are expired, but they are still listed as current*/


--Drop views that will be created in this query
DROP VIEW paying_but_expired;
DROP VIEW current_but_expired;


--First, create a view to show tenants who are paying rent or late fees, but have an expired contract
CREATE VIEW paying_but_expired AS
SELECT DISTINCT fk_tenant_code
FROM income a
```

```
INNER JOIN rental_contract b
ON a.fk_property_nickname = b.fk_property_nickname
WHERE (
        a.income_description = 'Rent'
        OR a.income_description = 'Late Fees'
      )
AND b.date_end < a.income_date;


--Create a view to show tenants who are  listed as current, but have an expired contract
CREATE VIEW current_but_expired AS
SELECT a.fk_tenant_code
FROM rental_contract a
WHERE a.current_tenant = 'Y'
AND a.date_end < (
                  SELECT sysdate FROM dual
                 );


--Merge the two views to create a comprehensive picture of tenants who need their contracts
renewed, including their names, their contract details,
--as well as the expiry date of their contract and the number of months the contract has been
expired for.


SELECT DISTINCT a.tenant_code, a.tenant_fn || ' ' || a.tenant_ln AS "Tenant name",
a.tenant_phone, a.tenant_email, d.date_end AS "Expiry Date",
```

```
    round(MONTHS_BETWEEN((select sysdate from dual), d.date_end)) AS "Months Expired"
FROM primary_tenant a
INNER JOIN rental_contract d
ON a.tenant_code = d.fk_tenant_code
INNER JOIN current_but_expired b
ON a.tenant_code = b.fk_tenant_code
FULL OUTER JOIN paying_but_expired c
ON a.tenant_code = c.fk_tenant_code
ORDER BY a.tenant_code ASC;

--Query 19: show the average income per property in each in descending order

--drop the views that will be created to complete this query
DROP VIEW properties_per_state;
DROP VIEW income_by_state;

--Create a view to show properties per state
CREATE VIEW properties_per_state AS
SELECT state, COUNT(property_nickname) AS "Number"
FROM property
GROUP BY state;

--Create a view to show income per state
CREATE VIEW income_by_state AS
```

```
SELECT b.state, SUM(a.income_amount) AS "State Income"
FROM income a
INNER JOIN property b
ON  a.fk_property_nickname = b.property_nickname
GROUP BY b.state;


--Show average income per property in each state
--Using the income_by_state and properties_per_state views
SELECT DISTINCT a.state, round(b."State Income"/c."Number",2) AS "Average Income Per Property"
FROM property a
INNER JOIN income_by_state b
ON a.state = b.state
INNER JOIN properties_per_state c
ON a.state = c.state
ORDER BY "Average Income Per Property" DESC;


--Query 20: For each zipcode that has at least one current tenant with a valid contract, show the
number of tenants in that zipcode.
SELECT a.zipcode, COUNT(c.fk_tenant_code) AS "Number of Tenants"
FROM property a
INNER JOIN rental_contract c
ON a.property_nickname = c.fk_property_nickname
INNER JOIN primary_tenant b
ON c.fk_tenant_code = b.tenant_code
```

```
WHERE b.tenant_code IN

(

   SELECT c.fk_tenant_code

   FROM rental_contract c

   WHERE c.date_end >

   (

      SELECT sysdate

      FROM dual

   )

)

AND c.current_tenant = 'Y'

GROUP BY a.zipcode;


--END
```

d. DDL, DML and Query Output

```
SQL> SET SERVEROUTPUT ON;

SQL> SET LINESIZE 150;

SQL> SET PAGESIZE 200;

SQL>

SQL> --Drops delete objects so you can run the same script multiple times

SQL>

SQL> --Drop Tables

SQL> DROP TABLE user_list CASCADE CONSTRAINTS;


Table USER_LIST dropped.


SQL> DROP TABLE database_transaction CASCADE CONSTRAINTS;


Table DATABASE_TRANSACTION dropped.


SQL> DROP TABLE primary_tenant CASCADE CONSTRAINTS;


Table PRIMARY_TENANT dropped.


SQL> DROP TABLE property CASCADE CONSTRAINTS;


Table PROPERTY dropped.
```

```
SQL> DROP TABLE rental_contract CASCADE CONSTRAINTS;

Table RENTAL_CONTRACT dropped.

SQL> DROP TABLE expense CASCADE CONSTRAINTS;

Table EXPENSE dropped.

SQL> DROP TABLE income CASCADE CONSTRAINTS;

Table INCOME dropped.

SQL>
SQL> --Drop sequences
SQL> DROP SEQUENCE user_code_Seq;

Sequence USER_CODE_SEQ dropped.

SQL> DROP SEQUENCE transaction_code_Seq;

Sequence TRANSACTION_CODE_SEQ dropped.

SQL> DROP SEQUENCE tenant_code_Seq;
```

```
Sequence TENANT_CODE_SEQ dropped.

SQL> DROP SEQUENCE expense_code_Seq;

Sequence EXPENSE_CODE_SEQ dropped.

SQL> DROP SEQUENCE income_code_Seq;

Sequence INCOME_CODE_SEQ dropped.

SQL> DROP SEQUENCE occupancy_code_Seq;

Sequence OCCUPANCY_CODE_SEQ dropped.

SQL> DROP SEQUENCE property_code_Seq;

Sequence PROPERTY_CODE_SEQ dropped.

SQL>
SQL> --Create the tables
SQL>
SQL> --Create user_list table
SQL> CREATE TABLE user_list
  2  (
```

```
  3   user_code         NUMERIC NOT NULL,
  4   security_level    VARCHAR (10) NOT NULL,
  5   username          VARCHAR (30),
  6   user_password     VARCHAR (10),
  7   user_email        VARCHAR(30),
  8   --This will ensure a UNIQUE primary key
  9   PRIMARY KEY (user_code)
 10  );

Table USER_LIST created.

SQL>
SQL> --Create database_transaction table
SQL> CREATE TABLE database_transaction
  2  (
  3   transaction_code          NUMERIC NOT NULL,
  4   entity                    VARCHAR (20) NOT NULL,
  5   entity_code               NUMERIC NOT NULL,
  6   transacted_by             VARCHAR(20) NOT NULL,
  7   transaction_date_time     TIMESTAMP(2) NOT NULL,
  8   transaction_description   VARCHAR(30),
  9   transaction_notes         VARCHAR(30),
 10   --This will ensure a UNIQUE primary key
 11   PRIMARY KEY (transaction_code)
```

```
 12  );
```

Table DATABASE_TRANSACTION created.

```
SQL>
SQL> --Create primary_tenant table
SQL> CREATE TABLE primary_tenant
  2  (
  3    tenant_code             NUMERIC NOT NULL,
  4    tenant_fn               VARCHAR (10) NOT NULL,
  5    tenant_ln               VARCHAR(10) NOT NULL,
  6    tenant_email            VARCHAR(30),
  7    tenant_phone             VARCHAR(20) NOT NULL,
  8    tenant_notes            VARCHAR(100),
  9    --This will ensire a UNIQUE primary key
 10     PRIMARY KEY (tenant_code)
 11  );
```

Table PRIMARY_TENANT created.

```
SQL>
SQL> --Create property table
SQL> CREATE TABLE property
  2  (
```

```
  3   property_code           NUMERIC NOT NULL,
  4   property_nickname       VARCHAR(50) NOT NULL,
  5   mortgage_years          DECIMAL (8,2) NOT NULL,
  6   property_address        VARCHAR(50) NOT NULL,
  7   state                   VARCHAR(2),
  8   zipcode                 VARCHAR (5),
  9   monthly_mortgage        DECIMAL(8,2) NOT NULL,
 10   hoa                     DECIMAL(8,2) NOT NULL,
 11   date_bought             DATE,
 12   sewage                  VARCHAR(30),
 13   electric                VARCHAR(30),
 14   property_notes          VARCHAR(100),
 15   --This will ensure a UNIQUE primary key
 16   PRIMARY KEY (property_nickname)
 17  );

Table PROPERTY created.

SQL>
SQL>
SQL> --Create rental_contract table
SQL> CREATE TABLE rental_contract
  2  (
  3   occupancy_code          NUMERIC NOT NULL,
```

```
 4   fk_tenant_code          NUMERIC NOT NULL,
 5   fk_property_nickname    VARCHAR(50) NOT NULL,
 6   rent                    DECIMAL(8,2) NOT NULL,
 7   charge_electric_to      VARCHAR(20),
 8   charge_sewage_to        VARCHAR(30),
 9   current_tenant          CHAR(1),
10   date_start              DATE NOT NULL,
11   date_end                DATE NOT NULL,
12   --This will ensure a UNIQUE primary key
13   PRIMARY KEY (occupancy_code),
14   FOREIGN KEY (fk_tenant_code) REFERENCES primary_tenant(tenant_code)
15 );

Table RENTAL_CONTRACT created.

SQL>
SQL> --Create expense table
SQL> CREATE TABLE expense
 2 (
 3   expense_code            NUMERIC NOT NULL,
 4   fk_property_nickname    VARCHAR(50) NOT NULL,
 5   expense_amount          DECIMAL(8,2) NOT NULL,
 6   expense_description     VARCHAR(50) NOT NULL,
 7   expense_date            DATE NOT NULL,
```

```
 8  expense_notes          VARCHAR(200),
 9  --This will ensure a UNIQUE primary key
10   PRIMARY KEY (expense_code),
11   FOREIGN KEY (fk_property_nickname) REFERENCES property(property_nickname)
12  );
```

Table EXPENSE created.

```
SQL>
SQL> --Create income table
SQL> CREATE TABLE income
 2  (
 3  income_code            NUMERIC NOT NULL,
 4  fk_property_nickname     VARCHAR(50) NOT NULL,
 5  income_amount          DECIMAL(8,2) NOT NULL,
 6  income_description       VARCHAR(50) NOT NULL,
 7  income_date            DATE NOT NULL,
 8  income_notes           VARCHAR(100),
 9  --This will ensure a UNIQUE primary key
10   PRIMARY KEY (income_code),
11   FOREIGN KEY (fk_property_nickname) REFERENCES property(property_nickname)
12  );
```

Table INCOME created.

```
SQL>

SQL> --Create sequences

SQL>

SQL> --Create property_code sequence

SQL> CREATE SEQUENCE  property_code_Seq

  2  START WITH 1

  3  INCREMENT BY 1;


Sequence PROPERTY_CODE_SEQ created.


SQL>

SQL> --Create user sequence

SQL> CREATE SEQUENCE user_code_Seq

  2  START WITH 100

  3  INCREMENT BY 1;


Sequence USER_CODE_SEQ created.


SQL>

SQL> --Create database_transcation sequence

SQL> CREATE SEQUENCE transaction_code_Seq

  2  START WITH 1

  3  INCREMENT BY 1;
```

```
Sequence TRANSACTION_CODE_SEQ created.

SQL>
SQL> --Create tenant_code sequence
SQL> CREATE SEQUENCE tenant_code_Seq
  2  START WITH 1
  3  INCREMENT BY 1;

Sequence TENANT_CODE_SEQ created.

SQL>
SQL> --Create occupancy_code sequence
SQL> CREATE SEQUENCE  occupancy_code_Seq
  2  START WITH 100
  3  INCREMENT BY 1;

Sequence OCCUPANCY_CODE_SEQ created.

SQL>
SQL> --Create expense_code sequence
SQL> CREATE SEQUENCE  expense_code_Seq
  2  START WITH 1
  3  INCREMENT BY 1;
```

```
Sequence EXPENSE_CODE_SEQ created.

SQL>
SQL> --Create income_code sequence
SQL> CREATE SEQUENCE  income_code_Seq
  2  START WITH 1
  3  INCREMENT BY 1;

Sequence INCOME_CODE_SEQ created.

SQL>
SQL> --Indexes
SQL>
SQL> --Create indexes for foreign keys
SQL>
SQL> --expense table
SQL> CREATE INDEX expense_property_nickname_index
  2  ON expense(fk_property_nickname);

Index EXPENSE_PROPERTY_NICKNAME_INDEX created.

SQL>
SQL> --income table
```

```
SQL> CREATE INDEX income_property_nickname_index
  2  ON income(fk_property_nickname);

Index INCOME_PROPERTY_NICKNAME_INDEX created.

SQL>
SQL> --rental_contract table
SQL> CREATE INDEX rental_tenant_index
  2  ON rental_contract(fk_tenant_code);

Index RENTAL_TENANT_INDEX created.

SQL>
SQL> CREATE INDEX rental_property_index
  2  ON rental_contract(fk_property_nickname);

Index RENTAL_PROPERTY_INDEX created.

SQL>
SQL> --Add some audit columns to the tables
SQL>
SQL> --user_list
SQL> ALTER TABLE user_list
  2  ADD (
```

```
  3  created_by VARCHAR2(30),
  4  date_created DATE,
  5  modified_by VARCHAR2(30),
  6  date_modified DATE
  7  );

Table USER_LIST altered.

SQL>
SQL> --property
SQL> ALTER TABLE property
  2  ADD (
  3  created_by VARCHAR2(30),
  4  date_created DATE,
  5  modified_by VARCHAR2(30),
  6  date_modified DATE
  7  );

Table PROPERTY altered.

SQL>
SQL> --primary_tenant
SQL> ALTER TABLE primary_tenant
  2  ADD (
```

```
  3  created_by VARCHAR2(30),
  4  date_created DATE,
  5  modified_by VARCHAR2(30),
  6  date_modified DATE
  7  );

Table PRIMARY_TENANT altered.

SQL>
SQL> --rental_contract
SQL> ALTER TABLE rental_contract
  2  ADD (
  3  created_by VARCHAR2(30),
  4  date_created DATE,
  5  modified_by VARCHAR2(30),
  6  date_modified DATE
  7  );

Table RENTAL_CONTRACT altered.

SQL>
SQL> --expense
SQL> ALTER TABLE expense
  2  ADD (
```

```
  3  created_by VARCHAR2(30),
  4  date_created DATE,
  5  modified_by VARCHAR2(30),
  6  date_modified DATE
  7  );

Table EXPENSE altered.

SQL>
SQL> --income
SQL> ALTER TABLE income
  2  ADD (
  3  created_by VARCHAR2(30),
  4  date_created DATE,
  5  modified_by VARCHAR2(30),
  6  date_modified DATE
  7  );

Table INCOME altered.

SQL>
SQL> --database_transaction table will not have audit columns as it is an audit table
SQL>
SQL> --Views
```

```
SQL>

SQL> --View each table (except database_transaction) without audit columns

SQL>

SQL> --user_list: this view shows basic user_list information without audit columns

SQL> CREATE OR REPLACE VIEW view_user_list AS

  2  SELECT user_code, security_level, username, user_password, user_email FROM user_list;


View VIEW_USER_LIST created.


SQL>

SQL> --property: this view shows basic property information without audit columns

SQL> CREATE OR REPLACE VIEW view_property AS

  2  SELECT property_code, property_nickname, mortgage_years, property_address,

monthly_mortgage, hoa, sewage, electric, property_notes FROM property;


View VIEW_PROPERTY created.


SQL>

SQL> --primary_tenant: this view shows basic primary_tenant information without audit columns

SQL> CREATE OR REPLACE VIEW view_primary_tenant AS

  2  SELECT tenant_code, tenant_fn, tenant_ln, tenant_email, tenant_phone, tenant_notes FROM

primary_tenant;


View VIEW_PRIMARY_TENANT created.
```

```
SQL>
SQL> --rental_contract: this view shows basic rental_contract information without audit columns
SQL> CREATE OR REPLACE VIEW view_rental_contract AS
  2  SELECT occupancy_code, fk_tenant_code, fk_property_nickname, rent, charge_electric_to,
charge_sewage_to, current_tenant, date_start, date_end FROM rental_contract;

View VIEW_RENTAL_CONTRACT created.

SQL>
SQL> --expense: this view shows basic expense information without audit columns
SQL> CREATE OR REPLACE VIEW view_expense AS
  2  SELECT expense_code, fk_property_nickname, expense_amount, expense_description,
expense_date, expense_notes FROM expense;

View VIEW_EXPENSE created.

SQL>
SQL> --income: this view shows basic income information without audit columns
SQL> CREATE OR REPLACE VIEW view_income AS
  2  SELECT income_code, fk_property_nickname, income_amount, income_description,
income_date, income_notes FROM income;

View VIEW_INCOME created.
```

```
SQL>
SQL> --Views that manipulate data to show helpful information to the property owner
SQL>
SQL> --Tenant phone view. This view is to create a phone list for the property owner to view and call
if necessary
SQL> CREATE OR REPLACE VIEW tenant_phone AS
  2  SELECT  tenant_fn || ' ' ||  tenant_ln AS "Name", tenant_phone AS "Phone Number" FROM
primary_tenant;

View TENANT_PHONE created.

SQL>
SQL> --Expenses by tenant occupancy view. The purpose of this view is to view all expenses incurred
by tenant. This is to see if a tenant is causing an abnormal amount of expenses due to damages
SQL> CREATE OR REPLACE VIEW tenant_expense AS
  2  SELECT c.tenant_fn || ' ' || c.tenant_ln AS "Name", a.fk_tenant_code AS "tenant_code",
b.expense_description, b.expense_amount
  3  FROM primary_tenant c
  4  INNER JOIN rental_contract a ON c.tenant_code = a.fk_tenant_code
  5  INNER JOIN expense b ON a.fk_property_nickname = b.fk_property_nickname;

View TENANT_EXPENSE created.
```

```
SQL>

SQL> --Income paid by each tenant view.

SQL> --First set the column size so that it displays in a readable fashion

SQL> column c1 heading "Name" format a15

SQL> column c2 heading "Property" format a15

SQL> column c3 heading "Paid On" format a15

SQL> column c4 heading "Description" format a15

SQL> column c5 heading "Amount" format a15

SQL> column c6 heading "Contract Dates" format a20

SQL>

SQL> --Create the view of income per tenant per property to see how much income has been
generated by each tenant per property

SQL> --This allows the property owner to see if rent payments have been missed, and if so, if late
fees were collected

SQL> CREATE OR REPLACE VIEW income_per_tenant_per_rental_contract AS

  2  SELECT a.tenant_fn || ' ' || a.tenant_ln c1, b.fk_property_nickname c2, c.income_date c3,
c.income_description c4, c.income_amount c5, b.date_start || ' ' || b.date_end c6

  3  FROM primary_tenant a

  4  INNER JOIN rental_contract b ON a.tenant_code = b.fk_tenant_code

  5  INNER JOIN income c ON b.fk_property_nickname = c.fk_property_nickname

  6  WHERE c.income_date <= (SELECT CURRENT_DATE FROM dual) AND c.income_date >=
b.date_start;


View INCOME_PER_TENANT_PER_RENTAL_CONTRACT created.
```

```
SQL>
SQL> --triggers
SQL>
SQL> --audit table database_transaction and sequence triggers for each table
SQL>
SQL> --property table
SQL>
SQL> --trigger to populate the database_transaction table when a row is inserted into property, as
well as to create a sequenced primary key when it is null
SQL> CREATE OR REPLACE TRIGGER property_insert_trig
  2  BEFORE INSERT
  3  ON property
  4  FOR EACH ROW
  5  DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
  6  BEGIN
  7  IF :NEW.property_code IS NULL THEN
  8  :NEW.property_code := property_code_Seq.NEXTVAL;
  9  END IF;
 10  entity := :NEW.property_code;
 11  SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
 12  SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
 13  INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
```

```
 14  VALUES (transaction_code_Seq.nextVal, 'property', entity, username, date_time, 'Property
added');
 15  END;
 16  /

Trigger PROPERTY_INSERT_TRIG compiled

SQL>
SQL> --trigger to populate the database_transaction table when a row is updated in property
SQL> CREATE OR REPLACE TRIGGER property_update_trig
  2  BEFORE UPDATE ON property
  3  FOR EACH ROW
  4  DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
  5  BEGIN
  6  IF :NEW.property_code IS NULL THEN
  7  :NEW.property_code := property_code_Seq.NEXTVAL;
  8  END IF;
  9  entity := :NEW.property_nickname;
 10  SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
 11  SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
 12  INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
 13  VALUES (transaction_code_Seq.nextVal, 'property', entity, username, date_time, 'Property
updated');
```

```
 14  END;
 15  /

Trigger PROPERTY_UPDATE_TRIG compiled


SQL>
SQL> --primary_tenant table
SQL>
SQL> --trigger to populate the database_transaction table when a row is inserted into
primary_tenant as well as to create a sequenced primary key when it is null
SQL> CREATE OR REPLACE TRIGGER primary_tenant_insert_trig
  2  BEFORE INSERT
  3  ON primary_tenant
  4  FOR EACH ROW
  5  DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
  6  BEGIN
  7  IF :NEW.tenant_code IS NULL THEN
  8  :NEW.tenant_code := tenant_code_Seq.NEXTVAL;
  9  END IF;
 10  entity := :NEW.tenant_code;
 11  SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
 12  SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
 13  INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
```

```
 14  VALUES (transaction_code_Seq.nextVal, 'primary_tenant', entity, username, date_time,
'Primary tenant added');
 15  END;
 16  /

Trigger PRIMARY_TENANT_INSERT_TRIG compiled

SQL>
SQL> --trigger to populate the database_transaction table when a row is updated in primary_tenant
table
SQL> CREATE OR REPLACE TRIGGER primary_tenant_update_trig
  2  AFTER UPDATE ON primary_tenant
  3  FOR EACH ROW
  4  DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
  5  BEGIN
  6  entity := :NEW.tenant_code;
  7  SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
  8  SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
  9  INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
 10  VALUES (transaction_code_Seq.nextVal, 'primary tenant', entity, username, date_time,
'Property updated');
 11  END;
 12  /
```

```
Trigger PRIMARY_TENANT_UPDATE_TRIG compiled


SQL>
SQL> --rental_contract table
SQL>
SQL> --trigger to populate the database_transaction table when a row is inserted into
rental_contract as well as to create a sequenced primary key when it is null
SQL> CREATE OR REPLACE TRIGGER rental_contract_insert_trig
  2  BEFORE INSERT
  3  ON rental_contract
  4  FOR EACH ROW
  5  DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
  6  BEGIN
  7  IF :NEW.occupancy_code IS NULL THEN
  8  :NEW.occupancy_code := occupancy_code_Seq.NEXTVAL;
  9  END IF;
 10  entity := :NEW.occupancy_code;
 11  SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
 12  SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
 13  INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
 14  VALUES (transaction_code_Seq.nextVal, 'Rental contract', entity, username, date_time, 'Rental
contract added');
```

```
 15  END;
 16  /

Trigger RENTAL_CONTRACT_INSERT_TRIG compiled

SQL>
SQL> --trigger to populate the database_transaction table when a row is updated in rental_contract
table
SQL> CREATE OR REPLACE TRIGGER rental_contract_update_trig
  2  AFTER UPDATE ON rental_contract
  3  FOR EACH ROW
  4  DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
  5  BEGIN
  6  entity := :NEW.occupancy_code;
  7  SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
  8  SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
  9  INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
 10  VALUES (transaction_code_Seq.nextVal, 'Rental contract', entity, username, date_time, 'Rental
contract updated');
 11  END;
 12  /

Trigger RENTAL_CONTRACT_UPDATE_TRIG compiled
```

```
SQL>
SQL> --expense table
SQL>
SQL> --trigger to populate the database_transaction table when a row is inserted into the expense
table as well as to create a sequenced primary key when it is null
SQL> CREATE OR REPLACE TRIGGER expense_insert_trig
  2  BEFORE INSERT
  3  ON expense
  4  FOR EACH ROW
  5  DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
  6  BEGIN
  7  IF :NEW.expense_code IS NULL THEN
  8  :NEW.expense_code := expense_code_Seq.NEXTVAL;
  9  END IF;
 10  entity := :NEW.expense_code;
 11  SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
 12  SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
 13  INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
 14  VALUES (transaction_code_Seq.nextVal, 'Expense', entity, username, date_time, 'Expense
added');
 15  END;
 16  /
```

```
Trigger EXPENSE_INSERT_TRIG compiled


SQL>
SQL> --trigger to populate the database_transaction table when a row is updated in expense table
SQL> CREATE OR REPLACE TRIGGER expense_update_trig
  2  AFTER UPDATE ON expense
  3  FOR EACH ROW
  4  DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
  5  BEGIN
  6  entity := :NEW.expense_code;
  7  SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
  8  SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
  9  INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
 10  VALUES (transaction_code_Seq.nextVal, 'expense', entity, username, date_time, 'expense
updated');
 11  END;
 12  /

Trigger EXPENSE_UPDATE_TRIG compiled


SQL>
SQL> --income table
```

```
SQL>
SQL> --trigger to populate the database_transaction table when a row is inserted into
rental_contract as well as to create a sequenced primary key when it is null
SQL> CREATE OR REPLACE TRIGGER income_insert_trig
  2  BEFORE INSERT
  3  ON income
  4  FOR EACH ROW
  5  DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
  6  BEGIN
  7  IF :NEW.income_code IS NULL THEN
  8  :NEW.income_code := income_code_Seq.NEXTVAL;
  9  END IF;
 10  entity := :NEW.income_code;
 11  SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
 12  SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
 13  INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
 14  VALUES (transaction_code_Seq.nextVal, 'Income', entity, username, date_time, 'Income
added');
 15  END;
 16  /

Trigger INCOME_INSERT_TRIG compiled
```

```
SQL>

SQL>

SQL> --trigger to populate the database_transaction table when a row is updated in income table

SQL> CREATE OR REPLACE TRIGGER income_update_trig

  2  AFTER UPDATE ON income

  3  FOR EACH ROW

  4  DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;

  5  BEGIN

  6  entity := :NEW.income_code;

  7  SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;

  8  SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;

  9  INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,

transaction_date_time, transaction_description)

 10  VALUES (transaction_code_Seq.nextVal, 'Income', entity, username, date_time, 'Income

updated');

 11  END;

 12  /


Trigger INCOME_UPDATE_TRIG compiled


SQL>

SQL> --user_list table

SQL>
```

```
SQL> --trigger to populate the database_transaction table when a row is inserted into user_list as
well as to create a sequenced primary key when it is null
SQL> CREATE OR REPLACE TRIGGER user_list_insert_trig
  2  BEFORE INSERT
  3  ON user_list
  4  FOR EACH ROW
  5  DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
  6  BEGIN
  7  IF :NEW.user_code IS NULL THEN
  8  :NEW.user_code := user_code_Seq.NEXTVAL;
  9  END IF;
 10  entity := :NEW.user_code;
 11  SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
 12  SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
 13  INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
 14  VALUES (transaction_code_Seq.nextVal, 'User', entity, username, date_time, 'User added');
 15  END;
 16  /

Trigger USER_LIST_INSERT_TRIG compiled

SQL>
```

```
SQL> --trigger to populate the database_transaction table when a row is updated in rental_contract
table
SQL> CREATE OR REPLACE TRIGGER user_list_update_trig
  2  AFTER UPDATE ON user_list
  3  FOR EACH ROW
  4  DECLARE entity NUMERIC; username VARCHAR(20); date_time TIMESTAMP;
  5  BEGIN
  6  entity := :NEW.user_code;
  7  SELECT sys_context('USERENV','CURRENT_USER') INTO username FROM dual;
  8  SELECT CURRENT_TIMESTAMP INTO date_time FROM dual;
  9  INSERT INTO database_transaction(transaction_code, entity, entity_code, transacted_by,
transaction_date_time, transaction_description)
 10  VALUES (transaction_code_Seq.nextVal, 'User', entity, username, date_time, 'User updated');
 11  END;
 12  /

Trigger USER_LIST_UPDATE_TRIG compiled

SQL>
SQL> --DML Script
SQL>
SQL> --insert some sample data
SQL>
SQL> --parent tables
```

```
SQL>

SQL> --primary_tenant table

SQL> INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email,

tenant_phone, tenant_notes)

  2  VALUES (tenant_code_Seq.nextVal, 'Lisa', 'Smith', 'lisasmith@gmail.com', '222 112 9090',

'requested to pay on the 5th of each month instead of the 1st');


1 row inserted.


SQL>

SQL> INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email,

tenant_phone, tenant_notes)

  2  VALUES (tenant_code_Seq.nextVal, 'Amy', 'Adams', 'amyadams@gmail.com', '222 111 1010',

'has a pet ferret');


1 row inserted.


SQL>

SQL> INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email,

tenant_phone, tenant_notes)

  2  VALUES (tenant_code_Seq.nextVal, 'Barbara', 'Bayleaf', 'barbay@gmail.com', '123 102 3090',

'requested the kitchen be repainted');


1 row inserted.
```

```
SQL>
SQL> INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email,
tenant_phone, tenant_notes)
  2  VALUES (tenant_code_Seq.nextVal, 'Curtis', 'Caveman', 'curtcave@gmail.com', '200 312 8766',
'drummer. may new to add soundproofing if neighbors complain');

1 row inserted.

SQL>
SQL> INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email,
tenant_phone, tenant_notes)
  2  VALUES (tenant_code_Seq.nextVal, 'Darlene', 'Davis', 'davisthedarling@gmail.com', '232 122
9990', 'single mom');

1 row inserted.

SQL>
SQL> INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email,
tenant_phone, tenant_notes)
  2  VALUES (tenant_code_Seq.nextVal, 'Elliot', 'Earl', 'earlelli@gmail.com', '213 100 0090',
'neighbors complain he is weird');

1 row inserted.
```

```
SQL>
SQL> INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email,
tenant_phone, tenant_notes)
  2  VALUES (tenant_code_Seq.nextVal, 'Farrah', 'Fawcett', 'farfaw@gmail.com', '122 000 9090', '2
dogs');

1 row inserted.

SQL>
SQL> INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email,
tenant_phone, tenant_notes)
  2  VALUES (tenant_code_Seq.nextVal, 'Gary', 'Indiana', 'onedollarhouse@gmail.com', '192 222
1190', 'inherited from previous owner');

1 row inserted.

SQL>
SQL> INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email,
tenant_phone, tenant_notes)
  2  VALUES (tenant_code_Seq.nextVal, 'Harry', 'Smith', 'harhar@gmail.com', '200 162 3489', 'reliable
tenant');

1 row inserted.
```

```
SQL>
SQL> INSERT INTO primary_tenant (tenant_code, tenant_fn, tenant_ln, tenant_email,
tenant_phone, tenant_notes)
  2  VALUES (tenant_code_Seq.nextVal, 'Ingrid', 'Indigo', 'indigo@gmail.com', '415 199 9090',
'repainted the livingroom purple. take out of deposit when she leaves');

1 row inserted.

SQL>
SQL> --user_list table
SQL> INSERT INTO user_list (user_code, security_level)
  2  VALUES (user_code_Seq.nextVal, 'tenant');

1 row inserted.

SQL>
SQL> INSERT INTO user_list (user_code, security_level)
  2  VALUES (user_code_Seq.nextVal, 'owner');

1 row inserted.

SQL>
SQL> INSERT INTO user_list (user_code, security_level)
```

```
  2  VALUES (user_code_Seq.nextVal, 'tenant');

1 row inserted.

SQL>
SQL> INSERT INTO user_list (user_code, security_level)
  2  VALUES (user_code_Seq.nextVal, 'owner');

1 row inserted.

SQL>
SQL> INSERT INTO user_list (user_code, security_level)
  2  VALUES (user_code_Seq.nextVal, 'tenant');

1 row inserted.

SQL>
SQL> INSERT INTO user_list (user_code, security_level)
  2  VALUES (user_code_Seq.nextVal, 'tenant');

1 row inserted.

SQL>
SQL> INSERT INTO user_list (user_code, security_level)
```

```
  2  VALUES (user_code_Seq.nextVal, 'tenant');

1 row inserted.

SQL>
SQL> INSERT INTO user_list (user_code, security_level)
  2  VALUES (user_code_Seq.nextVal, 'tenant');

1 row inserted.

SQL>
SQL> INSERT INTO user_list (user_code, security_level)
  2  VALUES (user_code_Seq.nextVal, 'tenant');

1 row inserted.

SQL>
SQL> INSERT INTO user_list (user_code, security_level)
  2  VALUES (user_code_Seq.nextVal, 'tenant');

1 row inserted.

SQL>
SQL> --property table
```

```
SQL>

SQL> INSERT INTO property (property_code, property_nickname, mortgage_years,

property_address, state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)

  2  VALUES (property_code_Seq.nextVal, 'Alaska', 30, '123 Alaska Street, Tacoma', 'WA', '98499',

900.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'Needs renovations');


1 row inserted.


SQL>

SQL> INSERT INTO property (property_code, property_nickname, mortgage_years,

property_address, state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)

  2  VALUES (property_code_Seq.nextVal, 'Edgewood Unit 1', 30, '123 Edgewood Street, Edgewood',

'MD', '21040', 700.00, 0.00, 'YuckYuck Sewage', 'Tacoma Electric', 'mortgage split with unit 2');


1 row inserted.


SQL>

SQL> INSERT INTO property (property_code, property_nickname, mortgage_years,

property_address, state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)

  2  VALUES (property_code_Seq.nextVal, 'Edgewood Unit 2', 30, '123 Edgewood Street, Edgewood',

'MD',  '21040', 700.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split with unit 1');


1 row inserted.
```

```
SQL>

SQL> INSERT INTO property (property_code, property_nickname, mortgage_years,

property_address, state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)

  2  VALUES (property_code_Seq.nextVal, 'Yakima Unit 1', 30, '123 Yakima Street, Overland Park',

'KS', '66204', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways');


1 row inserted.


SQL>

SQL> INSERT INTO property (property_code, property_nickname, mortgage_years,

property_address, state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)

  2  VALUES (property_code_Seq.nextVal, 'Yakima Unit 2', 30, '123 Yakima Street, Overland Park',

'KS', '66204', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways');


1 row inserted.


SQL>

SQL> INSERT INTO property (property_code, property_nickname, mortgage_years,

property_address,  state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)

  2  VALUES (property_code_Seq.nextVal, 'Yakima Unit 3', 30, '123 Yakima Street, Overland Park',

'KS', '66204', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways');


1 row inserted.
```

```
SQL>

SQL> INSERT INTO property (property_code, property_nickname, mortgage_years,
property_address,  state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
  2  VALUES (property_code_Seq.nextVal, 'Yakima Unit 4', 30, '123 Yakima Street, Overland Park',
'KS', '66204', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways');

1 row inserted.

SQL>

SQL> INSERT INTO property (property_code, property_nickname, mortgage_years,
property_address, state,  zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
  2  VALUES (property_code_Seq.nextVal, 'Monroe Unit 1', 30, '123 Monroe Street, Bel Air', 'MD',
'21014', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways. high-end
renovation in 2021');

1 row inserted.

SQL>

SQL> INSERT INTO property (property_code, property_nickname, mortgage_years,
property_address,  state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
  2  VALUES (property_code_Seq.nextVal, 'Monroe Unit 2', 30, '123 Monroe Street, Bel Air', 'MD',
'21014', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways');

1 row inserted.
```

```
SQL>
SQL> INSERT INTO property (property_code, property_nickname, mortgage_years,
property_address, state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
  2  VALUES (property_code_Seq.nextVal, 'Monroe Unit 3', 30, '123 Monroe Street, Bel Air', 'MD',
'21014', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways. Recently
renovated in 2022');

1 row inserted.

SQL>
SQL> INSERT INTO property (property_code, property_nickname, mortgage_years,
property_address, state, zipcode, monthly_mortgage, hoa, sewage, electric, property_notes)
  2  VALUES (property_code_Seq.nextVal, 'Monroe Unit 4', 30, '123 Monroe Street, Bel Air', 'MD',
'21014', 500.00, 0.00, 'Tacoma Sewage', 'Tacoma Electric', 'mortgage split 4 ways. undergoing
renovation');

1 row inserted.

SQL>
SQL> --child tables
SQL>
SQL> --rental_contract table
```

```
SQL> INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
  2  VALUES (occupancy_code_seq.nextval, 1, 'Alaska', 1100.00, 'Tenant', 'Tenant', 'Y', DATE '2020-
01-01', DATE '2020-12-31');

1 row inserted.

SQL>
SQL> INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
  2  VALUES (occupancy_code_seq.nextval, 2, 'Edgewood Unit 1', 800.00, 'Tenant', 'Tenant', 'N', DATE
'2020-01-01', DATE '2020-12-31');

1 row inserted.

SQL>
SQL> INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
  2  VALUES (occupancy_code_seq.nextval, 2, 'Edgewood Unit 1', 900.00, 'Tenant', 'Tenant', 'N', DATE
'2021-01-01', DATE '2021-12-31');

1 row inserted.

SQL>
```

```
SQL> INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
  2  VALUES (occupancy_code_seq.nextval, 2, 'Edgewood Unit 1', 1100.00, 'Tenant', 'Tenant', 'Y',
DATE '2022-01-01', DATE '2022-12-31');

1 row inserted.

SQL>
SQL> INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
  2  VALUES (occupancy_code_seq.nextval, 3, 'Edgewood Unit 2', 950.00, 'Tenant', 'Tenant', 'Y', DATE
'2022-01-01', DATE '2022-12-31');

1 row inserted.

SQL>
SQL> INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
  2  VALUES (occupancy_code_seq.nextval, 4, 'Yakima Unit 1', 1100.00, 'Tenant', 'Tenant', 'Y', DATE
'2022-01-01', DATE '2022-12-31');

1 row inserted.

SQL>
```

```
SQL> INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
  2  VALUES (occupancy_code_seq.nextval, 5, 'Yakima Unit 2', 550.00, 'Tenant', 'Tenant', 'Y', DATE
'2022-01-01', DATE '2022-12-31');

1 row inserted.

SQL>
SQL> INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
  2  VALUES (occupancy_code_seq.nextval, 6, 'Yakima Unit 3', 1800.00, 'Tenant', 'Tenant', 'Y', DATE
'2022-01-01', DATE '2022-12-31');

1 row inserted.

SQL>
SQL> INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
  2  VALUES (occupancy_code_seq.nextval, 7, 'Yakima Unit 4', 1150.00, 'Tenant', 'Tenant', 'Y', DATE
'2022-01-01', DATE '2022-12-31');

1 row inserted.

SQL>
```

```
SQL> INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
  2  VALUES (occupancy_code_seq.nextval, 8, 'Monroe Unit 1', 12800.00, 'Tenant', 'Tenant', 'Y', DATE
'2022-01-01', DATE '2022-12-31');

1 row inserted.

SQL>
SQL> INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
  2  VALUES (occupancy_code_seq.nextval, 9, 'Monroe Unit 2', 800.00, 'Tenant', 'Tenant', 'Y', DATE
'2020-01-01', DATE '2020-12-31');

1 row inserted.

SQL>
SQL> INSERT INTO rental_contract(occupancy_code, fk_tenant_code, fk_property_nickname, rent,
charge_electric_to, charge_sewage_to, current_tenant, date_start, date_end)
  2  VALUES (occupancy_code_seq.nextval, 10, 'Monroe Unit 3', 900.00, 'Tenant', 'Tenant', 'Y', DATE
'2020-01-01', DATE '2020-12-31');

1 row inserted.

SQL>
```

```
SQL> --expense table
SQL> INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
  2  VALUES (expense_code_Seq.nextval, 'Monroe Unit 2', 150.00, 'Blocked Drain', DATE '2020-03-
12', 'Hair in shower drain');

1 row inserted.

SQL>
SQL> INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
  2  VALUES (expense_code_Seq.nextval, 'Alaska', 100.00, 'Door handles', DATE '2020-03-12', 'tenant
is afraid of doorknobs');

1 row inserted.

SQL>
SQL> INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
  2  VALUES (expense_code_Seq.nextval, 'Edgewood Unit 1', 2450.00, 'Exterminator', DATE '2022-05-
12', 'infestation due to secret litter of ferret pups in the drywall');

1 row inserted.
```

```
SQL>
SQL> INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
  2  VALUES (expense_code_Seq.nextval, 'Edgewood Unit 2', 1150.00, 'Paint and Painter', DATE
'2022-09-12', 'Kitchen repainted due to old cooking stains on wall');

1 row inserted.

SQL>
SQL> INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
  2  VALUES (expense_code_Seq.nextval, 'Edgewood Unit 2', 2150.00, 'Exterminator', DATE '2022-08-
01', 'Ferrets found in wood panelling. Flea infestation due to this. Ferrets taken to Unit 1 to their
mother');

1 row inserted.

SQL>
SQL> INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
  2  VALUES (expense_code_Seq.nextval, 'Edgewood Unit 2', 1550.00, 'Wall repair', DATE '2020-07-
27', 'Hole between Unit 1 and Unit 2 discovered');

1 row inserted.
```

```
SQL>
SQL> INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
  2  VALUES (expense_code_Seq.nextval, 'Monroe Unit 1', 350.00, 'Chocolate fountain repair', DATE
'2022-03-12', 'Chocolate fountain in bathroom needed a new pipe');

1 row inserted.

SQL>
SQL> INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
  2  VALUES (expense_code_Seq.nextval, 'Monroe Unit 1', 250.00, 'Chandeliar Reshining', DATE
'2022-05-12', 'Chandelier needed reshining/repolishing');

1 row inserted.

SQL>
SQL> INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
  2  VALUES (expense_code_Seq.nextval, 'Monroe Unit 1', 100.00, 'Blocked Drain', DATE '2022-03-
12', 'Diamonds from toilet seat dislodged into drain. Needed retrival');

1 row inserted.
```

```
SQL>
SQL> INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
  2  VALUES (expense_code_Seq.nextval, 'Monroe Unit 1', 550.00, 'Tank cleaning', DATE '2022-05-
12', 'Jellyfish tank in the lobby required annual cleaning');

1 row inserted.

SQL>
SQL> INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
  2  VALUES (expense_code_Seq.nextval, 'Alaska', 850.00, 'New carpets', DATE '2022-02-02', 'new
carpets installed');

1 row inserted.

SQL>
SQL> INSERT INTO expense(expense_code, fk_property_nickname, expense_amount,
expense_description, expense_date, expense_notes)
  2  VALUES (expense_code_Seq.nextval, 'Yakima Unit 1', 1850.00, 'Porch repair', DATE '2022-09-24',
'porch was rotting and needed replacement');

1 row inserted.
```

```
SQL>
SQL> --income table
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Alaska', 1100.00, 'Rent', DATE '2022-10-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Alaska', 1100.00, 'Rent', DATE '2022-09-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Alaska', 1100.00, 'Rent', DATE '2022-08-01');

1 row inserted.

SQL>
```

```
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Alaska', 1100.00, 'Rent', DATE '2022-07-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Alaska', 1100.00, 'Rent', DATE '2022-06-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Alaska', 1100.00, 'Rent', DATE '2022-05-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Alaska', 1100.00, 'Rent', DATE '2022-02-01');
```

```
1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Monroe Unit 1', 12800.00, 'Rent', DATE '2022-01-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Monroe Unit 1', 12800.00, 'Rent', DATE '2022-02-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Monroe Unit 1', 12800.00, 'Rent', DATE '2022-03-01');

1 row inserted.
```

```
SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Monroe Unit 1', 100.00, 'Late fee', DATE '2022-04-15');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Monroe Unit 1', 100.00, 'Late fee', DATE '2022-05-15');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Monroe Unit 1', 38400.00, 'Rents due to date', DATE '2022-
06-01');

1 row inserted.

SQL>
```

```
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-01-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-02-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-03-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-04-01');
```

```
1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-05-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-06-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 100.00, 'Late Fee', DATE '2022-06-10');

1 row inserted.
```

```
SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-06-25');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-07-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-08-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
```

```
  2  VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-09-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Edgewood Unit 2', 950.00, 'Rent', DATE '2022-10-01');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Monroe Unit 3', 100.00, 'Late Fee', DATE '2022-01-10');

1 row inserted.

SQL>
SQL> INSERT INTO income(income_code, fk_property_nickname, income_amount,
income_description, income_date)
  2  VALUES (income_code_Seq.nextval, 'Yakima Unit 3', 100.00, 'Late Fee', DATE '2022-01-10');

1 row inserted.
```

```
SQL>

SQL>

SQL>

SQL> --database_transaction not needed to be added as this is done automatically via trigger

SQL>

SQL> --Database queries to show that objects were created

SQL>

SQL> --table queries (but not other tables in the dbst connection)

SQL>

SQL> --show the tables exist

SQL> SELECT table_name FROM user_tables WHERE sample_size IS NULL;


TABLE_NAME

------------------------------------------------------------------------------------------------------------------------

INCOME

RENTAL_CONTRACT

DATABASE_TRANSACTION

EXPENSE

PRIMARY_TENANT

USER_LIST

PROPERTY


7 rows selected.
```

```
SQL>
SQL> --Describe tables
SQL> --user_list table
SQL> DESCRIBE user_list;
Name          Null?    Type
----------- -------- ------------
USER_CODE      NOT NULL NUMBER(38)
SECURITY_LEVEL NOT NULL VARCHAR2(10)
USERNAME              VARCHAR2(30)
USER_PASSWORD         VARCHAR2(10)
USER_EMAIL            VARCHAR2(30)
CREATED_BY            VARCHAR2(30)
DATE_CREATED          DATE
MODIFIED_BY           VARCHAR2(30)
DATE_MODIFIED         DATE
SQL>
SQL> --property table
SQL> DESCRIBE property;
Name            Null?    Type
----------------- -------- -------------
PROPERTY_CODE     NOT NULL NUMBER(38)
PROPERTY_NICKNAME NOT NULL VARCHAR2(50)
MORTGAGE_YEARS    NOT NULL NUMBER(8,2)
```

```
PROPERTY_ADDRESS  NOT NULL VARCHAR2(50)

STATE                  VARCHAR2(2)

ZIPCODE                VARCHAR2(5)

MONTHLY_MORTGAGE  NOT NULL NUMBER(8,2)

HOA             NOT NULL NUMBER(8,2)

DATE_BOUGHT             DATE

SEWAGE                  VARCHAR2(30)

ELECTRIC                VARCHAR2(30)

PROPERTY_NOTES          VARCHAR2(100)

CREATED_BY              VARCHAR2(30)

DATE_CREATED            DATE

MODIFIED_BY             VARCHAR2(30)

DATE_MODIFIED           DATE

SQL>

SQL> --primary_tenant table

SQL> DESCRIBE primary_tenant;

Name         Null?    Type

------------- -------- -------------

TENANT_CODE   NOT NULL NUMBER(38)

TENANT_FN     NOT NULL VARCHAR2(10)

TENANT_LN     NOT NULL VARCHAR2(10)

TENANT_EMAIL          VARCHAR2(30)

TENANT_PHONE  NOT NULL VARCHAR2(20)

TENANT_NOTES          VARCHAR2(100)
```

```
CREATED_BY           VARCHAR2(30)

DATE_CREATED         DATE

MODIFIED_BY          VARCHAR2(30)

DATE_MODIFIED        DATE

SQL>

SQL> --rental_contract table

SQL> DESCRIBE rental_contract;

Name              Null?    Type

----------------- -------- ------------

OCCUPANCY_CODE       NOT NULL NUMBER(38)

FK_TENANT_CODE       NOT NULL NUMBER(38)

FK_PROPERTY_NICKNAME NOT NULL VARCHAR2(50)

RENT              NOT NULL NUMBER(8,2)

CHARGE_ELECTRIC_TO          VARCHAR2(20)

CHARGE_SEWAGE_TO            VARCHAR2(30)

CURRENT_TENANT              CHAR(1)

DATE_START          NOT NULL DATE

DATE_END           NOT NULL DATE

CREATED_BY                VARCHAR2(30)

DATE_CREATED               DATE

MODIFIED_BY                VARCHAR2(30)

DATE_MODIFIED              DATE

SQL>

SQL> --expense table
```

```
SQL> DESCRIBE expense;
Name              Null?    Type
----------------- -------- -------------
EXPENSE_CODE        NOT NULL NUMBER(38)
FK_PROPERTY_NICKNAME NOT NULL VARCHAR2(50)
EXPENSE_AMOUNT      NOT NULL NUMBER(8,2)
EXPENSE_DESCRIPTION  NOT NULL VARCHAR2(50)
EXPENSE_DATE        NOT NULL DATE
EXPENSE_NOTES               VARCHAR2(200)
CREATED_BY                  VARCHAR2(30)
DATE_CREATED                DATE
MODIFIED_BY                 VARCHAR2(30)
DATE_MODIFIED               DATE
SQL>
SQL> --income table
SQL> DESCRIBE income;
Name              Null?    Type
----------------- -------- -------------
INCOME_CODE         NOT NULL NUMBER(38)
FK_PROPERTY_NICKNAME NOT NULL VARCHAR2(50)
INCOME_AMOUNT       NOT NULL NUMBER(8,2)
INCOME_DESCRIPTION   NOT NULL VARCHAR2(50)
INCOME_DATE         NOT NULL DATE
INCOME_NOTES                VARCHAR2(100)
```

```
CREATED_BY                VARCHAR2(30)

DATE_CREATED              DATE

MODIFIED_BY               VARCHAR2(30)

DATE_MODIFIED             DATE

SQL>

SQL> --database_transaction table

SQL> DESCRIBE database_transaction;

Name                 Null?    Type

---------------------- -------- ------------

TRANSACTION_CODE       NOT NULL NUMBER(38)

ENTITY                 NOT NULL VARCHAR2(20)

ENTITY_CODE            NOT NULL NUMBER(38)

TRANSACTED_BY          NOT NULL VARCHAR2(20)

TRANSACTION_DATE_TIME   NOT NULL TIMESTAMP(2)

TRANSACTION_DESCRIPTION         VARCHAR2(30)

TRANSACTION_NOTES               VARCHAR2(30)

SQL>

SQL> --indexes

SQL> --show indexes exist

SQL> -- only objects created after 7 March are selected due to the STUDENT database objects

showing up as well

SQL> COLUMN a1 HEADING "Object Name" FORMAT a35

SQL> COLUMN a2 HEADING "Object Type" FORMAT a15

SQL> COLUMN a3 HEADING "Status" FORMAT a15
```

```
SQL> COLUMN a4 HEADING "Created" FORMAT a15

SQL>

SQL> SELECT object_name a1, object_type a2,  status a3, created a4

  2  FROM user_objects

  3  WHERE created > '07-MAR-22'

  4  AND GENERATED = 'N'

  5  AND object_type = 'INDEX' ;


Object Name                    Object Type     Status         Created

---------------------------------- --------------- --------------- ---------------

EXPENSE_PROPERTY_NICKNAME_INDEX     INDEX          VALID          01-NOV-22

INCOME_PROPERTY_NICKNAME_INDEX      INDEX          VALID          01-NOV-22

RENTAL_TENANT_INDEX                 INDEX          VALID          01-NOV-22

RENTAL_PROPERTY_INDEX               INDEX          VALID          01-NOV-22


SQL>

SQL> --triggers

SQL> --view triggers (selected only object_name to simplify viewing)

SQL> COLUMN b1 HEADING "Object Name" FORMAT a30

SQL> COLUMN b2 HEADING "Object Type" FORMAT a15

SQL> COLUMN b3 HEADING "Status" FORMAT a15

SQL> COLUMN b4 HEADING "Created" FORMAT a15

SQL>

SQL> SELECT object_name b1, object_type b2,  status b3, created b4
```

```
 2  FROM user_objects
 3  WHERE object_type='TRIGGER';

Object Name                    Object Type     Status         Created
------------------------------ --------------- -------------- ---------------
EXPENSE_UPDATE_TRIG            TRIGGER         VALID          01-NOV-22
INCOME_INSERT_TRIG             TRIGGER         VALID          01-NOV-22
INCOME_UPDATE_TRIG             TRIGGER         VALID          01-NOV-22
USER_LIST_INSERT_TRIG          TRIGGER         VALID          01-NOV-22
EXPENSE_INSERT_TRIG            TRIGGER         VALID          01-NOV-22
RENTAL_CONTRACT_UPDATE_TRIG    TRIGGER         VALID          01-NOV-22
USER_LIST_UPDATE_TRIG          TRIGGER         VALID          01-NOV-22
PRIMARY_TENANT_INSERT_TRIG     TRIGGER         VALID          01-NOV-22
PRIMARY_TENANT_UPDATE_TRIG     TRIGGER         VALID          01-NOV-22
PROPERTY_INSERT_TRIG           TRIGGER         VALID          01-NOV-22
PROPERTY_UPDATE_TRIG           TRIGGER         VALID          01-NOV-22
RENTAL_CONTRACT_INSERT_TRIG    TRIGGER         VALID          01-NOV-22

12 rows selected.

SQL>
SQL> --view views
SQL> --selected only object_name to simplify viewing
SQL> COLUMN d1 HEADING "Object Name" FORMAT a40
```

```
SQL> COLUMN d2 HEADING "Object Type" FORMAT a15

SQL> COLUMN d3 HEADING "Status" FORMAT a15

SQL> COLUMN d4 HEADING "Created" FORMAT a15

SQL>

SQL> SELECT object_name d1, object_type d2,  status d3, created d4

  2  FROM user_objects

  3  WHERE object_type='VIEW';


Object Name                     Object Type     Status        Created

--------------------------------------- --------------- --------------- ---------------

TENANT_PHONE                    VIEW          VALID         05-OCT-22

TENANT_EXPENSE                  VIEW          VALID         06-OCT-22

STUDENTCITY                     VIEW          VALID         21-OCT-22

MONTHS_OF_RENT                  VIEW          INVALID       01-NOV-22

RENT_DUE                        VIEW          INVALID       01-NOV-22

DUE_AND_RECEIVED_RENT           VIEW          INVALID       01-NOV-22

EXPIRED_BUT_PAYING              VIEW          INVALID       29-OCT-22

PAYING_BUT_EXPIRED              VIEW          INVALID       01-NOV-22

CURRENT_BUT_EXPIRED             VIEW          INVALID       01-NOV-22

PROPERTIES_PER_STATE            VIEW          INVALID       01-NOV-22

INCOME_BY_STATE                 VIEW          INVALID       01-NOV-22

AVERAGE_PER_ZIP                 VIEW          INVALID       28-OCT-22

INCOME_PER_TENANT_PER_RENTAL_CONTRACT   VIEW          VALID         08-OCT-22

VIEW_USER_LIST                  VIEW          VALID         08-OCT-22
```

```
VIEW_PROPERTY                  VIEW         VALID        08-OCT-22
VIEW_PRIMARY_TENANT            VIEW         VALID        08-OCT-22
VIEW_RENTAL_CONTRACT           VIEW         VALID        08-OCT-22
VIEW_EXPENSE                   VIEW         VALID        08-OCT-22
VIEW_INCOME                    VIEW         VALID        08-OCT-22
ACCUMULATED_EXPENSE            VIEW         INVALID      28-OCT-22
EXPENSE_SUMMARY                VIEW         INVALID      01-NOV-22
TENANT_PER_ZIP                 VIEW         INVALID      28-OCT-22
CURRENT_OCCUPANCY              VIEW         VALID        01-NOV-22

23 rows selected.

SQL>
SQL> --view indexes
SQL> --only object__name selected to simplify viewing, and only objects created after 7 March are
selected due to the STUDENT database objects showing up as well
SQL> COLUMN e1 HEADING "Object Name" FORMAT a20
SQL> COLUMN e2 HEADING "Object Type" FORMAT a15
SQL> COLUMN e3 HEADING "Status" FORMAT a15
SQL> COLUMN e4 HEADING "Created" FORMAT a15
SQL> SELECT object_name e1, object_type e2,  status e3, created e4 FROM user_objects WHERE
object_type='SEQUENCE' AND created > '07-MAR-22';

Object Name         Object Type     Status         Created
```

```
-------------------- --------------- --------------- ---------------
PROPERTY_CODE_SEQ    SEQUENCE        VALID           01-NOV-22
USER_CODE_SEQ        SEQUENCE        VALID           01-NOV-22
TRANSACTION_CODE_SEQ SEQUENCE        VALID           01-NOV-22
TENANT_CODE_SEQ      SEQUENCE        VALID           01-NOV-22
OCCUPANCY_CODE_SEQ   SEQUENCE        VALID           01-NOV-22
EXPENSE_CODE_SEQ     SEQUENCE        VALID           01-NOV-22
INCOME_CODE_SEQ      SEQUENCE        VALID           01-NOV-22

7 rows selected.

SQL>
SQL>
SQL> --Query 1: Select all columns and all rows from one table
SQL> SELECT *
  2  FROM primary_tenant;

TENANT_CODE TENANT_FN  TENANT_LN  TENANT_EMAIL                  TENANT_PHONE
----------- ---------- ---------- ----------------------------- --------------------
TENANT_NOTES                                                                                     CREATED_BY
DATE_CREA
---------------------------------------------------------------------------------------------------- ------------------------------ --
-------
MODIFIED_BY                    DATE_MODI
```

------------------------------ ---------

1 Lisa Smith lisasmith@gmail.com 222 112 9090

requested to pay on the 5th of each month instead of the 1st

2 Amy Adams amyadams@gmail.com 222 111 1010

has a pet ferret

3 Barbara Bayleaf barbay@gmail.com 123 102 3090

requested the kitchen be repainted

4 Curtis Caveman curtcave@gmail.com 200 312 8766

drummer. may new to add soundproofing if neighbors complain

5 Darlene Davis davisthedarling@gmail.com 232 122 9990

single mom

6 Elliot Earl earlelli@gmail.com 213 100 0090

neighbors complain he is weird

7 Farrah Fawcett farfaw@gmail.com 122 000 9090

2 dogs

8 Gary Indiana onedollarhouse@gmail.com 192 222 1190

inherited from previous owner

9 Harry Smith harhar@gmail.com 200 162 3489

reliable tenant

10 Ingrid Indigo indigo@gmail.com 415 199 9090

repainted the livingroom purple. take out of deposit when she leaves

10 rows selected.

SQL>

SQL> --Query 2: Select five columns and all rows from one table

SQL> SELECT fk_property_nickname, expense_amount, expense_description, expense_date,

expense_notes

```
 2  FROM expense;

FK_PROPERTY_NICKNAME                EXPENSE_AMOUNT EXPENSE_DESCRIPTION

EXPENSE_D

--------------------------------------------------- -------------- ------------------------------------------------- ---------

EXPENSE_NOTES

-------------------------------------------------------------------------------------------------------------------------------------

-----------------

Monroe Unit 2                       150 Blocked Drain                   12-MAR-20

Hair in shower drain


Alaska                              100 Door handles                    12-MAR-20

tenant is afraid of doorknobs


Edgewood Unit 1                     2450 Exterminator                   12-MAY-22

infestation due to secret litter of ferret pups in the drywall


Edgewood Unit 2                     1150 Paint and Painter              12-SEP-22

Kitchen repainted due to old cooking stains on wall


Edgewood Unit 2                     2150 Exterminator                   01-AUG-22

Ferrets found in wood panelling. Flea infestation due to this. Ferrets taken to Unit 1 to their mother


Edgewood Unit 2                     1550 Wall repair                    27-JUL-20
```

Hole between Unit 1 and Unit 2 discovered

Monroe Unit 1 350 Chocolate fountain repair 12-MAR-22

Chocolate fountain in bathroom needed a new pipe

Monroe Unit 1 250 Chandeliar Reshining 12-MAY-22

Chandelier needed reshining/repolishing

Monroe Unit 1 100 Blocked Drain 12-MAR-22

Diamonds from toilet seat dislodged into drain. Needed retrival

Monroe Unit 1 550 Tank cleaning 12-MAY-22

Jellyfish tank in the lobby required annual cleaning

Alaska 850 New carpets 02-FEB-22

new carpets installed

Yakima Unit 1 1850 Porch repair 24-SEP-22

porch was rotting and needed replacement

12 rows selected.

SQL>

```
SQL> --Query 3: Select all columns from all rows from one view

SQL> SELECT *

  2  FROM  income_per_tenant_per_rental_contract;
```

Name	Property	Paid On	Description	Amount	Contract Dates
Lisa Smith	Alaska	01-OCT-22	Rent	1100	01-JAN-20 31-DEC-20
Lisa Smith	Alaska	01-SEP-22	Rent	1100	01-JAN-20 31-DEC-20
Lisa Smith	Alaska	01-AUG-22	Rent	1100	01-JAN-20 31-DEC-20
Lisa Smith	Alaska	01-JUL-22	Rent	1100	01-JAN-20 31-DEC-20
Lisa Smith	Alaska	01-JUN-22	Rent	1100	01-JAN-20 31-DEC-20
Lisa Smith	Alaska	01-MAY-22	Rent	1100	01-JAN-20 31-DEC-20
Lisa Smith	Alaska	01-FEB-22	Rent	1100	01-JAN-20 31-DEC-20
Barbara Bayleaf	Edgewood Unit 2	01-JAN-22	Rent	950	01-JAN-22 31-DEC-22
Barbara Bayleaf	Edgewood Unit 2	01-FEB-22	Rent	950	01-JAN-22 31-DEC-22
Barbara Bayleaf	Edgewood Unit 2	01-MAR-22	Rent	950	01-JAN-22 31-DEC-22
Barbara Bayleaf	Edgewood Unit 2	01-APR-22	Rent	950	01-JAN-22 31-DEC-22
Barbara Bayleaf	Edgewood Unit 2	01-MAY-22	Rent	950	01-JAN-22 31-DEC-22
Barbara Bayleaf	Edgewood Unit 2	01-JUN-22	Rent	950	01-JAN-22 31-DEC-22
Barbara Bayleaf	Edgewood Unit 2	10-JUN-22	Late Fee	100	01-JAN-22 31-DEC-22
Barbara Bayleaf	Edgewood Unit 2	25-JUN-22	Rent	950	01-JAN-22 31-DEC-22
Barbara Bayleaf	Edgewood Unit 2	01-JUL-22	Rent	950	01-JAN-22 31-DEC-22
Barbara Bayleaf	Edgewood Unit 2	01-AUG-22	Rent	950	01-JAN-22 31-DEC-22
Barbara Bayleaf	Edgewood Unit 2	01-SEP-22	Rent	950	01-JAN-22 31-DEC-22

Barbara Bayleaf Edgewood Unit 2 01-OCT-22 Rent 950 01-JAN-22 31-DEC-22

Elliot Earl Yakima Unit 3 10-JAN-22 Late Fee 100 01-JAN-22 31-DEC-22

Gary Indiana Monroe Unit 1 01-JAN-22 Rent 12800 01-JAN-22 31-DEC-22

Gary Indiana Monroe Unit 1 01-FEB-22 Rent 12800 01-JAN-22 31-DEC-22

Gary Indiana Monroe Unit 1 01-MAR-22 Rent 12800 01-JAN-22 31-DEC-22

Gary Indiana Monroe Unit 1 15-APR-22 Late fee 100 01-JAN-22 31-DEC-22

Gary Indiana Monroe Unit 1 15-MAY-22 Late fee 100 01-JAN-22 31-DEC-22

Gary Indiana Monroe Unit 1 01-JUN-22 Rents due to da 38400 01-JAN-22 31-DEC-22

te

Ingrid Indigo Monroe Unit 3 10-JAN-22 Late Fee 100 01-JAN-20 31-DEC-20

27 rows selected.

SQL>

SQL> --Query 4: Using a join on 2 tables, select all columns and all rows from the tables without the use of a Cartesian product

SQL> SELECT *

2 FROM rental_contract

3 INNER JOIN primary_tenant

4 ON rental_contract.fk_tenant_code = primary_tenant.tenant_code;

OCCUPANCY_CODE FK_TENANT_CODE FK_PROPERTY_NICKNAME RENT

CHARGE_ELECTRIC_TO CHARGE_SEWAGE_TO C

-------------- -------------- --- ---------- ------------------- ---------------------

--------- -

DATE_STAR DATE_END CREATED_BY DATE_CREA ODIFIED_BY DATE_MODI

TENANT_CODE TENANT_FN TENANT_LN

--------- --------- ----------------------------- --------- ----------------------------- --------- ----------- ---------- ----------

TENANT_EMAIL TENANT_PHONE

----------------------------- --------------------

TENANT_NOTES CREATED_BY

DATE_CREA

-- ------------------------------ --

MODIFIED_BY DATE_MODI

----------------------------- ---------

100 1 Alaska 1100 Tenant Tenant Y

01-JAN-20 31-DEC-20 1 Lisa Smith

lisasmith@gmail.com 222 112 9090

requested to pay on the 5th of each month instead of the 1st

101 2 Edgewood Unit 1 800 Tenant Tenant

N

01-JAN-20 31-DEC-20 2 Amy Adams

amyadams@gmail.com 222 111 1010

has a pet ferret

102 2 Edgewood Unit 1 900 Tenant Tenant

N

01-JAN-21 31-DEC-21 2 Amy Adams

amyadams@gmail.com 222 111 1010

has a pet ferret

103 2 Edgewood Unit 1 1100 Tenant Tenant

Y

01-JAN-22 31-DEC-22 2 Amy Adams

amyadams@gmail.com 222 111 1010

has a pet ferret

104 3 Edgewood Unit 2 950 Tenant Tenant

Y

01-JAN-22 31-DEC-22 3 Barbara Bayleaf

barbay@gmail.com 123 102 3090

requested the kitchen be repainted

105 4 Yakima Unit 1 1100 Tenant Tenant

Y

01-JAN-22 31-DEC-22 4 Curtis Caveman

curtcave@gmail.com 200 312 8766

drummer. may new to add soundproofing if neighbors complain

106 5 Yakima Unit 2 550 Tenant Tenant Y

01-JAN-22 31-DEC-22 5 Darlene Davis

davisthedarling@gmail.com 232 122 9990

single mom

107 6 Yakima Unit 3 1800 Tenant Tenant

Y

01-JAN-22 31-DEC-22 6 Elliot Earl

earlelli@gmail.com 213 100 0090

neighbors complain he is weird

108 7 Yakima Unit 4 1150 Tenant Tenant

Y

01-JAN-22 31-DEC-22 7 Farrah Fawcett

farfaw@gmail.com 122 000 9090

2 dogs

109 8 Monroe Unit 1 12800 Tenant Tenant

Y

01-JAN-22 31-DEC-22 8 Gary Indiana

onedollarhouse@gmail.com 192 222 1190

inherited from previous owner

110 9 Monroe Unit 2 800 Tenant Tenant

Y

01-JAN-20 31-DEC-20 9 Harry Smith

harhar@gmail.com 200 162 3489

reliable tenant

111 10 Monroe Unit 3 900 Tenant Tenant

Y

01-JAN-20 31-DEC-20 10 Ingrid Indigo

indigo@gmail.com 415 199 9090

repainted the livingroom purple. take out of deposit when she leaves

12 rows selected.

```
SQL>
SQL> --Query 5: Select and order data retrieved from one table
SQL> --First set the column size so that it displays in a readable fashion
SQL> column c1 heading "Expense Code" format a10
SQL> column c2 heading "Property" format a20
SQL> column c3 heading "Expense Amount" format a10
SQL> column c4 heading "Description" format a20
SQL> column c5 heading "Date" format a10
SQL> column c6 heading "Notes" format a30
SQL>
SQL> SELECT expense_code c1, fk_property_nickname c2, expense_amount c3, expense_description
c4, expense_date c5, expense_notes c6
  2  FROM expense
  3  ORDER BY expense_date ASC;

Expense Co Property           Expense Am Description         Date      Notes
---------- -------------------- ---------- -------------------- ---------- ------------------------------
         1 Monroe Unit 2               150 Blocked Drain        12-MAR-20  Hair in shower drain
         2 Alaska                      100 Door handles         12-MAR-20  tenant is afraid of doorknobs
         6 Edgewood Unit 2            1550 Wall repair          27-JUL-20  Hole between Unit 1 and Unit 2
                                                                           discovered
```

11 Alaska	850 New carpets	02-FEB-22	new carpets installed
9 Monroe Unit 1	100 Blocked Drain	12-MAR-22	Diamonds from toilet seat dislodged into drain. Needed retrival
7 Monroe Unit 1	350 Chocolate fountain repair	12-MAR-22	Chocolate fountain in bathroom needed a new pipe
8 Monroe Unit 1	250 Chandeliar Reshining	12-MAY-22	Chandelier needed reshining/repolishing
10 Monroe Unit 1	550 Tank cleaning	12-MAY-22	Jellyfish tank in the lobby required annual cleaning
3 Edgewood Unit 1	2450 Exterminator	12-MAY-22	infestation due to secret litter of ferret pups in the drywall
5 Edgewood Unit 2	2150 Exterminator	01-AUG-22	Ferrets found in wood panelling. Flea infestation due to this. Ferrets taken to Unit 1 to their mother

```
         4 Edgewood Unit 2          1150 Paint and Painter    12-SEP-22  Kitchen repainted due to old c
                                                                          ooking stains on wall

        12 Yakima Unit 1            1850 Porch repair         24-SEP-22  porch was rotting and needed r
                                                                          eplacement


12 rows selected.


SQL>
SQL> --Query 6: Using a join on 3 tables, select 5 columns from the 3 tables. Use syntax that would
limit the output to 10 rows
SQL> SELECT c.property_nickname AS "Property", b.tenant_fn AS "First name", b.tenant_ln AS "Last
name", a.rent AS "Rent", a.date_end AS "Contract End"
  2  FROM rental_contract a
  3  INNER JOIN primary_tenant b
  4  ON a.fk_tenant_code = b.tenant_code
  5  INNER JOIN property c
  6  ON c.property_nickname = a.fk_property_nickname
  7  WHERE b.tenant_code < 11;

Property                       First name Last name       Rent Contract
------------------------------------------------------ ---------- ---------- ---------- ---------
Alaska                         Lisa       Smith          1100 31-DEC-20
```

```
Edgewood Unit 1                 Amy       Adams          800 31-DEC-20

Edgewood Unit 1                 Amy       Adams          900 31-DEC-21

Edgewood Unit 1                 Amy       Adams         1100 31-DEC-22

Edgewood Unit 2                 Barbara   Bayleaf        950 31-DEC-22

Yakima Unit 1                   Curtis    Caveman       1100 31-DEC-22

Yakima Unit 2                   Darlene   Davis          550 31-DEC-22

Yakima Unit 3                   Elliot    Earl          1800 31-DEC-22

Yakima Unit 4                   Farrah    Fawcett       1150 31-DEC-22

Monroe Unit 1                   Gary      Indiana      12800 31-DEC-22

Monroe Unit 2                   Harry     Smith          800 31-DEC-20

Monroe Unit 3                   Ingrid    Indigo         900 31-DEC-20


12 rows selected.


SQL>

SQL> --Query 7: Select distinct rows using joins on 3 tables

SQL> SELECT DISTINCT b.sewage, b.property_nickname AS "Properties", c.tenant_code AS "Tenant

Code"

  2  FROM rental_contract a

  3  INNER JOIN property b

  4  ON a.fk_property_nickname = b.property_nickname

  5  INNER JOIN primary_tenant c

  6  ON a.fk_tenant_code = c.tenant_code;
```

SEWAGE	Properties	Tenant Code
Tacoma Sewage	Yakima Unit 1	4
YuckYuck Sewage	Edgewood Unit 1	2
Tacoma Sewage	Monroe Unit 2	9
Tacoma Sewage	Edgewood Unit 2	3
Tacoma Sewage	Yakima Unit 3	6
Tacoma Sewage	Yakima Unit 4	7
Tacoma Sewage	Yakima Unit 2	5
Tacoma Sewage	Monroe Unit 1	8
Tacoma Sewage	Alaska	1
Tacoma Sewage	Monroe Unit 3	10

10 rows selected.

```
SQL>
SQL> --Query 8: Use GROUP BY and HAVING in a select statement using one or more tables
SQL> SELECT b.sewage, COUNT(b.property_nickname) AS "Number of Properties",
COUNT(c.tenant_code) AS "Number of Contracts"
  2  FROM rental_contract a
  3  INNER JOIN property b
  4  ON a.fk_property_nickname = b.property_nickname
  5  INNER JOIN primary_tenant c
  6  ON a.fk_tenant_code = c.tenant_code
```

```
  7  GROUP BY b.sewage
  8  HAVING COUNT(b.property_nickname) >3;

SEWAGE                    Number of Properties Number of Contracts
------------------------------ ------------------- -------------------
Tacoma Sewage                              9                   9

SQL>
SQL> --Query 9: Use IN clause to select data from one or more tables
SQL> SELECT a.tenant_fn, a.tenant_ln, b.fk_tenant_code, c.property_nickname
  2  FROM primary_tenant a
  3  INNER JOIN rental_contract b
  4  ON a.tenant_code = b.fk_tenant_code
  5  INNER JOIN property c
  6  ON b.fk_property_nickname = c.property_nickname
  7  WHERE a.tenant_code
  8  IN ('1' , '3', '9');

TENANT_FN  TENANT_LN  FK_TENANT_CODE PROPERTY_NICKNAME
---------- ---------- -------------- --------------------------------------------------
Lisa       Smith                1 Alaska
Barbara    Bayleaf              3 Edgewood Unit 2
Harry      Smith                9 Monroe Unit 2
```

```
SQL>

SQL> --Query 10: Select length of one column from one table

SQL> SELECT LENGTH (tenant_notes) AS "Length of Tenant Notes"

  2  FROM primary_tenant;


Length of Tenant Notes

----------------------

                60

                16

                34

                59

                10

                30

                 6

                29

                15

                68


10 rows selected.


SQL>

SQL> --Query 11: Delete one record from one table. Use select statements to demonstrate the table

contents before and after the DELETE statement.

SQL> --Make sure you use ROLLBACK afterwards so that the data will not be physically removed
```

```
SQL>

SQL> --Show the table before the DELETE statement

SQL> --First set the column size so that it displays in a readable fashion

SQL> column c1 heading "Expense Code" format a10

SQL> column c2 heading "Property" format a20

SQL> column c3 heading "Expense Amount" format a10

SQL> column c4 heading "Description" format a20

SQL> column c5 heading "Date" format a10

SQL> column c6 heading "Notes" format a30

SQL> SELECT expense_code c1, fk_property_nickname c2, expense_amount c3, expense_description
c4, expense_date c5, expense_notes c6 FROM expense;


Expense Co Property           Expense Am Description        Date      Notes
---------- -------------------- ---------- -------------------- ---------- ------------------------------
         1 Monroe Unit 2               150 Blocked Drain       12-MAR-20  Hair in shower drain

         2 Alaska                      100 Door handles        12-MAR-20  tenant is afraid of doorknobs

         3 Edgewood Unit 1            2450 Exterminator        12-MAY-22  infestation due to secret litt
                                           er of ferret pups in the drywa
                                           ll


         4 Edgewood Unit 2            1150 Paint and Painter   12-SEP-22  Kitchen repainted due to old c
                                           ooking stains on wall


         5 Edgewood Unit 2            2150 Exterminator        01-AUG-22  Ferrets found in wood panellin
```

g. Flea infestation due to thi

s. Ferrets taken to Unit 1 to

their mother

6 Edgewood Unit 2 1550 Wall repair 27-JUL-20 Hole between Unit 1 and Unit 2 discovered

7 Monroe Unit 1 350 Chocolate fountain r 12-MAR-22 Chocolate fountain in bathroom epair needed a new pipe

8 Monroe Unit 1 250 Chandeliar Reshining 12-MAY-22 Chandelier needed reshining/re polishing

9 Monroe Unit 1 100 Blocked Drain 12-MAR-22 Diamonds from toilet seat disl odged into drain. Needed retri

val

10 Monroe Unit 1 550 Tank cleaning 12-MAY-22 Jellyfish tank in the lobby re quired annual cleaning

11 Alaska 850 New carpets 02-FEB-22 new carpets installed

12 Yakima Unit 1 1850 Porch repair 24-SEP-22 porch was rotting and needed r eplacement

```
12 rows selected.


SQL>

SQL>

SQL> --Delete one record

SQL> DELETE

  2  FROM expense

  3  WHERE expense_description = 'Wall repair';


1 row deleted.


SQL>

SQL> --Show the table after the DELETE statement

SQL> --First set the column size so that it displays in a readable fashion

SQL> column c1 heading "Expense Code" format a10

SQL> column c2 heading "Property" format a20

SQL> column c3 heading "Expense Amount" format a10

SQL> column c4 heading "Description" format a20

SQL> column c5 heading "Date" format a10

SQL> column c6 heading "Notes" format a30

SQL> SELECT expense_code c1, fk_property_nickname c2, expense_amount c3, expense_description

c4, expense_date c5, expense_notes c6 FROM expense;
```

Expense Co	Property	Expense Am	Description	Date	Notes
1	Monroe Unit 2	150	Blocked Drain	12-MAR-20	Hair in shower drain
2	Alaska	100	Door handles	12-MAR-20	tenant is afraid of doorknobs
3	Edgewood Unit 1	2450	Exterminator	12-MAY-22	infestation due to secret litter of ferret pups in the drywall
4	Edgewood Unit 2	1150	Paint and Painter	12-SEP-22	Kitchen repainted due to old cooking stains on wall
5	Edgewood Unit 2	2150	Exterminator	01-AUG-22	Ferrets found in wood panelling. Flea infestation due to this. Ferrets taken to Unit 1 to their mother
7	Monroe Unit 1	350	Chocolate fountain repair	12-MAR-22	Chocolate fountain in bathroom needed a new pipe
8	Monroe Unit 1	250	Chandeliar Reshining	12-MAY-22	Chandelier needed reshining/repolishing
9	Monroe Unit 1	100	Blocked Drain	12-MAR-22	Diamonds from toilet seat dislodged into drain. Needed retri

```
val

10 Monroe Unit 1            550 Tank cleaning       12-MAY-22  Jellyfish tank in the lobby re
                                                               quired annual cleaning

11 Alaska                   850 New carpets         02-FEB-22  new carpets installed
12 Yakima Unit 1            1850 Porch repair       24-SEP-22  porch was rotting and needed r
                                                               eplacement


11 rows selected.

SQL>
SQL>
SQL> --Rollback after DELETE statement
SQL> --Hwoever, the rollback rolls back more than just the DELETE statement (my insert statements,
etc.) and causes issues when the script is run all together
SQL> --Hence, the rollback is commented out and must be uncommented, run individually (just for
the delete statement), and recommented to allow the entire scritp to be run without errors
SQL> --ROLLBACK;
SQL>
SQL>
SQL> -- Query 12: Update one record from one table. Use select statements to demonstrate the
table contents before and after the UPDATE statement.
```

```
SQL> --Make sure you use ROLLBACK afterwards so that the data will not be physically removed
SQL>
SQL> --Show table contents before update
SQL> --First set the column size so that it displays in a readable fashion
SQL> column c1 heading "Expense Code" format a10
SQL> column c2 heading "Property" format a20
SQL> column c3 heading "Expense Amount" format a10
SQL> column c4 heading "Description" format a20
SQL> column c5 heading "Date" format a10
SQL> column c6 heading "Notes" format a30
SQL> SELECT expense_code c1, fk_property_nickname c2, expense_amount c3, expense_description
c4, expense_date c5, expense_notes c6 FROM expense;

Expense Co Property           Expense Am Description        Date      Notes
---------- -------------------- ---------- -------------------- ---------- ------------------------------
         1 Monroe Unit 2               150 Blocked Drain       12-MAR-20  Hair in shower drain
         2 Alaska                      100 Door handles        12-MAR-20  tenant is afraid of doorknobs
         3 Edgewood Unit 1            2450 Exterminator        12-MAY-22  infestation due to secret litt
                                           er of ferret pups in the drywa
                                           ll

         4 Edgewood Unit 2            1150 Paint and Painter   12-SEP-22  Kitchen repainted due to old c
                                           ooking stains on wall
```

5	Edgewood Unit 2	2150	Exterminator	01-AUG-22	Ferrets found in wood panellin g. Flea infestation due to thi s. Ferrets taken to Unit 1 to their mother
7	Monroe Unit 1	350	Chocolate fountain r epair	12-MAR-22	Chocolate fountain in bathroom needed a new pipe
8	Monroe Unit 1	250	Chandeliar Reshining	12-MAY-22	Chandelier needed reshining/re polishing
9	Monroe Unit 1	100	Blocked Drain	12-MAR-22	Diamonds from toilet seat disl odged into drain. Needed retri val
10	Monroe Unit 1	550	Tank cleaning	12-MAY-22	Jellyfish tank in the lobby re quired annual cleaning
11	Alaska	850	New carpets	02-FEB-22	new carpets installed
12	Yakima Unit 1	1850	Porch repair	24-SEP-22	porch was rotting and needed r eplacement

11 rows selected.

```
SQL>
SQL> --Update the table
SQL> UPDATE expense
  2  SET expense_description = 'Ferret Whisperer'
  3  WHERE expense_code = '5';

1 row updated.

SQL>
SQL> --Show table after update
SQL> --First set the column size so that it displays in a readable fashion
SQL> column c1 heading "Expense Code" format a10
SQL> column c2 heading "Property" format a20
SQL> column c3 heading "Expense Amount" format a10
SQL> column c4 heading "Description" format a20
SQL> column c5 heading "Date" format a10
SQL> column c6 heading "Notes" format a30
SQL> SELECT expense_code c1, fk_property_nickname c2, expense_amount c3, expense_description
c4, expense_date c5, expense_notes c6 FROM expense;

Expense Co Property           Expense Am Description         Date       Notes
---------- -------------------- ---------- -------------------- ---------- ------------------------------
         1 Monroe Unit 2               150 Blocked Drain        12-MAR-20  Hair in shower drain
```

2 Alaska 100 Door handles 12-MAR-20 tenant is afraid of doorknobs

3 Edgewood Unit 1 2450 Exterminator 12-MAY-22 infestation due to secret litt
er of ferret pups in the drywa
ll

4 Edgewood Unit 2 1150 Paint and Painter 12-SEP-22 Kitchen repainted due to old c
ooking stains on wall

5 Edgewood Unit 2 2150 Ferret Whisperer 01-AUG-22 Ferrets found in wood panellin
g. Flea infestation due to thi
s. Ferrets taken to Unit 1 to
their mother

7 Monroe Unit 1 350 Chocolate fountain r 12-MAR-22 Chocolate fountain in bathroom
epair needed a new pipe

8 Monroe Unit 1 250 Chandeliar Reshining 12-MAY-22 Chandelier needed reshining/re
polishing

9 Monroe Unit 1 100 Blocked Drain 12-MAR-22 Diamonds from toilet seat disl
odged into drain. Needed retri
val

10 Monroe Unit 1 550 Tank cleaning 12-MAY-22 Jellyfish tank in the lobby re

quired annual cleaning

11 Alaska 850 New carpets 02-FEB-22 new carpets installed

12 Yakima Unit 1 1850 Porch repair 24-SEP-22 porch was rotting and needed r

eplacement

11 rows selected.

SQL>

SQL> --Rollback the update

SQL> --(however, this undoes my table inserts as well, renderign the rest of this script problematic hence, this rollback is commented out.)

SQL> --When using the update, it should be uncommented and rolled back individually (not the whole script)

SQL> --ROLLBACK;

SQL>

SQL> --Perform 8 Additional Advanced Queries

SQL>

SQL> --Query 13: Calculate the minimum, maximum and average expense per zipcode

SQL> SELECT b.zipcode, MIN(a.expense_amount) AS "Min Expense", MAX(a.expense_amount) AS "Max Expense", ROUND(AVG(a.expense_amount)) AS "Avg Expense"

  2 FROM expense a

  3 INNER JOIN property b

```
  4  ON a.fk_property_nickname = b.property_nickname
  5  GROUP BY b.zipcode;

ZIPCO Min Expense Max Expense Avg Expense
----- ----------- ----------- -----------
21040        1150        2450        1917
66204        1850        1850        1850
21014         100         550         280
98499         100         850         475

SQL>
SQL> --Query 14: List the current occupants
SQL>
SQL> --Drop the view that will be created in this query
SQL> DROP VIEW current_occupancy;

View CURRENT_OCCUPANCY dropped.

SQL>
SQL> --Create a view of the current occupants with valid contracts (valid in the current year) with
the property name and the dates of the contract
SQL> --This view is quite useful, and thus will be used in some queries that follow, to reduce
redundant work
SQL> CREATE VIEW current_occupancy AS
```

```
  2  SELECT DISTINCT d.property_nickname, d.zipcode, b.tenant_code,  b.tenant_fn || ' ' ||
b.tenant_ln AS "Tenant", c.date_start, c.date_end
  3  FROM primary_tenant b
  4  INNER JOIN rental_contract c
  5  ON b.tenant_code = c.fk_tenant_code
  6  INNER JOIN property d
  7  ON d.property_nickname = c.fk_property_nickname
  8  WHERE c.current_tenant = 'Y'
  9  AND c.date_end > (
 10                SELECT add_months(
 11                                  (
 12                                    SELECT sysdate
 13                                    FROM dual
 14                                  ), 12*-1
 15                                 ) FROM dual
 16               );

View CURRENT_OCCUPANCY created.


SQL>
SQL> --Show the current occupants
SQL> SELECT *
  2  FROM current_occupancy;
```

```
PROPERTY_NICKNAME                       ZIPCO TENANT_CODE Tenant            DATE_STAR
DATE_END
---------------------------------------------------- ----- ----------- -------------------- --------- ---------
Monroe Unit 1                           21014         8 Gary Indiana        01-JAN-22 31-DEC-22
Edgewood Unit 1                         21040         2 Amy Adams           01-JAN-22 31-DEC-22
Yakima Unit 2                           66204         5 Darlene Davis       01-JAN-22 31-DEC-22
Yakima Unit 4                           66204         7 Farrah Fawcett      01-JAN-22 31-DEC-22
Edgewood Unit 2                         21040         3 Barbara Bayleaf     01-JAN-22 31-DEC-22
Yakima Unit 1                           66204         4 Curtis Caveman      01-JAN-22 31-DEC-22
Yakima Unit 3                           66204         6 Elliot Earl         01-JAN-22 31-DEC-22

7 rows selected.


SQL>
SQL> --Query 15: Show the average expense per zipcode in the last year-to-date
SQL> SELECT DISTINCT a.zipcode, ROUND(AVG(b.expense_amount)) AS "Avg Expense"
  2  FROM property a
  3  FULL JOIN expense b
  4  ON a.property_nickname = b.fk_property_nickname
  5  WHERE b.expense_date >= (SELECT TO_CHAR(ADD_MONTHS((SELECT sysdate FROM dual), 12*-
1),'dd-MON-yyyy') FROM dual)
  6  AND b.expense_date <= (SELECT SYSDATE FROM DUAL)
  7  GROUP BY a.zipcode
  8  ORDER BY "Avg Expense" DESC;
```

```
ZIPCO Avg Expense
----- -----------
21040       1917
66204       1850
98499        850
21014        313


SQL>
SQL> --Query 16: Show tenants that are accumulating more than the average expenses in their
zipcode in the last year
SQL> --We will use the previously developed view "current_occupancy"
SQL>
SQL> --Drop the view that will be created in this query
SQL> DROP VIEW expense_summary;


View EXPENSE_SUMMARY dropped.


SQL>
SQL> --first, calculate the expected versus actual expediture during the life of the tenant's contract
(not year to date, nor calendar year, but each contract year)
SQL> CREATE VIEW expense_summary AS
```

```
  2  SELECT a.tenant_code, a.zipcode, NVL(ROUND(MONTHS_BETWEEN((SELECT sysdate FROM dual),
a.date_start)/12, 2) * b."Avg Expense", 0) AS "Expected", NVL(c."Expenses Accumulated",0) AS
"Actual"
  3  FROM current_occupancy a
  4  INNER JOIN average_per_zip b
  5  ON a.zipcode = b.zipcode
  6  FULL JOIN accumulated_expense c
  7  ON c.tenant_code = a.tenant_code;

View EXPENSE_SUMMARY created.

SQL>
SQL> ---Next, show the tenants that accumulated more expenses than what was expected/average
using the cumulative view "expense summary"  developed in this query
SQL> SELECT a.tenant_code, a.tenant_fn || ' ' || a.tenant_ln AS "Tenant", a.tenant_phone,
b."Actual" - b."Expected" AS "Over Average Per ZIP By"
  2  FROM primary_tenant a
  3  INNER JOIN expense_summary b
  4  ON a.tenant_code = b.tenant_code
  5  WHERE b."Actual" > b."Expected";

TENANT_CODE Tenant              TENANT_PHONE       Over Average Per ZIP By
----------- ---------------------- -------------------- ------------------------
          2 Amy Adams           222 111 1010                    858.89
```

```
3 Barbara Bayleaf       123 102 3090                    1708.89

4 Curtis Caveman        200 312 8766                     314.5

8 Gary Indiana          192 222 1190                    1017.6


SQL>

SQL> --Query 16: Calculate how much income per tenant has been accumulated from properties in
Washington state and Maryland

SQL> --We will use the previously developed view "current_occupancy" to assist

SQL>

SQL> SELECT a.property_nickname, a."Tenant", a.tenant_code, SUM(b.income_amount) AS "Income
Accumulated"

  2  FROM current_occupancy a

  3  INNER JOIN income b

  4  ON a.property_nickname = b.fk_property_nickname

  5  INNER JOIN property c

  6  ON a.property_nickname = c.property_nickname

  7  WHERE c.state IN ('WA', 'MD')

  8  GROUP BY a.property_nickname, a."Tenant", a.tenant_code;


PROPERTY_NICKNAME                  Tenant            TENANT_CODE Income Accumulated

-------------------------------------------------- -------------------- ----------- ------------------

Monroe Unit 1                      Gary Indiana                8           77000

Edgewood Unit 2                    Barbara Bayleaf             3           10550
```

```
SQL>

SQL>

SQL> --Query 17: List the tenants with current rental contracts who are behind in rent payments.

SQL> --We will use previously developed view, "current_occupancy", and develop new views based

off of this to further the queries

SQL> DROP VIEW months_of_rent;


View MONTHS_OF_RENT dropped.


SQL> DROP VIEW rent_due;


View RENT_DUE dropped.


SQL> DROP VIEW due_and_received_rent;


View DUE_AND_RECEIVED_RENT dropped.


SQL>

SQL> --First, create a view to calculate the number of months of rent due for contracts that are

current

SQL> CREATE VIEW months_of_rent AS

  2  SELECT tenant_code, round (

  3                       MONTHS_BETWEEN(

  4                               (
```

```
 5                                SELECT sysdate
 6                                FROM dual
 7                               ), date_start
 8                              )
 9                      ) AS "Rent Payments Due"
10  FROM current_occupancy;

View MONTHS_OF_RENT created.

SQL>
SQL> --Calculate the dollar amount of rent due over the course of the rental contract to date per
tenant for current contracts, using the views created in this query
SQL> CREATE VIEW rent_due AS
  2  SELECT a.tenant_code, a."Rent Payments Due" * b.rent AS "Rent Due"
  3  FROM months_of_rent a
  4  INNER JOIN rental_contract b
  5  ON a.tenant_code = b.fk_tenant_code
  6  WHERE b.date_end  > (
  7                   SELECT sysdate FROM dual
  8                  );

View RENT_DUE created.

SQL>
```

```
SQL> --Create a view showing the rent due and the rent received per current occupant, using the
views created in this query to this point
SQL> CREATE VIEW due_and_received_rent AS
  2  SELECT a.tenant_code, c."Rent Due", SUM(b.income_amount) AS "Rent Received"
  3  FROM current_occupancy a
  4  RIGHT OUTER JOIN rent_due c
  5  ON a.tenant_code = c.tenant_code
  6  INNER JOIN rental_contract d
  7  ON c.tenant_code = d.fk_tenant_code
  8  INNER JOIN property e
  9  ON d.fk_property_nickname = e.property_nickname
 10  INNER JOIN income b
 11  ON b.fk_property_nickname = e.property_nickname
 12  WHERE b.income_date BETWEEN TO_DATE(a.date_start) AND TO_DATE(a.date_end)
 13  AND b.income_description = 'Rent'
 14  GROUP BY a.tenant_code,  c."Rent Due";

View DUE_AND_RECEIVED_RENT created.

SQL>
SQL> --Show the details of the tenants with current rental contracts who are behind in rent, using
the cumulative view created previously in this query
SQL> SELECT a.tenant_fn, a.tenant_ln, a.tenant_code, a.tenant_phone, a.tenant_email, ABS(b."Rent
Received" - b."Rent Due") AS "Overdue By"
```

```
  2  FROM primary_tenant a
  3  RIGHT OUTER JOIN due_and_received_rent b
  4  ON a.tenant_code = b.tenant_code
  5  WHERE b."Rent Due" > b."Rent Received";

TENANT_FN  TENANT_LN  TENANT_CODE TENANT_PHONE        TENANT_EMAIL                Overdue
By
---------- ---------- ---------- ------------------- ------------------------------ ----------
Gary       Indiana           8 192 222 1190        onedollarhouse@gmail.com          89600

SQL>
SQL> /*Query 18: List tenants who need their rental contracts renewed.
SQL>This includes tenants do  not have current rental contracts, but are still paying rent or late fees,
i.e. are still actively living in the unit,
SQL>as well as tenants whose contracts are expired, but they are still listed as current*/
SQL>
SQL> --Drop views that will be created in this query
SQL> DROP VIEW paying_but_expired;

View PAYING_BUT_EXPIRED dropped.

SQL> DROP VIEW current_but_expired;

View CURRENT_BUT_EXPIRED dropped.
```

```
SQL>
SQL> --First, create a view to show tenants who are paying rent or late fees, but have an expired
contract
SQL> CREATE VIEW paying_but_expired AS
  2  SELECT DISTINCT fk_tenant_code
  3  FROM income a
  4  INNER JOIN rental_contract b
  5  ON a.fk_property_nickname = b.fk_property_nickname
  6  WHERE (
  7          a.income_description = 'Rent'
  8          OR a.income_description = 'Late Fees'
  9        )
 10  AND b.date_end < a.income_date;

View PAYING_BUT_EXPIRED created.

SQL>
SQL> --Create a view to show tenants who are  listed as current, but have an expired contract
SQL> CREATE VIEW current_but_expired AS
  2  SELECT a.fk_tenant_code
  3  FROM rental_contract a
  4  WHERE a.current_tenant = 'Y'
  5  AND a.date_end < (
```

```
  6                 SELECT sysdate FROM dual
  7                 );
```

View CURRENT_BUT_EXPIRED created.

```
SQL>
SQL> --Merge the two views to create a comprehensive picture of tenants who need their contracts
renewed, including their names, their contract details,
SQL> --as well as the expiry date of their contract and the number of months the contract has been
expired for.
SQL>
SQL> SELECT DISTINCT a.tenant_code, a.tenant_fn || ' ' || a.tenant_ln AS "Tenant name",
a.tenant_phone, a.tenant_email, d.date_end AS "Expiry Date",
  2       round(MONTHS_BETWEEN((select sysdate from dual), d.date_end)) AS "Months Expired"
  3  FROM primary_tenant a
  4  INNER JOIN rental_contract d
  5  ON a.tenant_code = d.fk_tenant_code
  6  INNER JOIN current_but_expired b
  7  ON a.tenant_code = b.fk_tenant_code
  8  FULL OUTER JOIN paying_but_expired c
  9  ON a.tenant_code = c.fk_tenant_code
 10  ORDER BY a.tenant_code ASC;
```

```
TENANT_CODE Tenant name         TENANT_PHONE       TENANT_EMAIL              Expiry Da
Months Expired
----------- -------------------- ------------------- ------------------------------ --------- --------------
          1 Lisa Smith           222 112 9090       lisasmith@gmail.com            31-DEC-20           22
          9 Harry Smith          200 162 3489       harhar@gmail.com               31-DEC-20           22
         10 Ingrid Indigo        415 199 9090       indigo@gmail.com               31-DEC-20           22

SQL>
SQL> --Query 19: show the average income per property in each in descending order
SQL>
SQL> --drop the views that will be created to complete this query
SQL> DROP VIEW properties_per_state;

View PROPERTIES_PER_STATE dropped.

SQL> DROP VIEW income_by_state;

View INCOME_BY_STATE dropped.

SQL>
SQL> --Create a view to show properties per state
SQL> CREATE VIEW properties_per_state AS
  2  SELECT state, COUNT(property_nickname) AS "Number"
  3  FROM property
```

```
  4  GROUP BY state;


View PROPERTIES_PER_STATE created.


SQL>
SQL> --Create a view to show income per state
SQL> CREATE VIEW income_by_state AS
  2  SELECT b.state, SUM(a.income_amount) AS "State Income"
  3  FROM income a
  4  INNER JOIN property b
  5  ON  a.fk_property_nickname = b.property_nickname
  6  GROUP BY b.state;


View INCOME_BY_STATE created.


SQL>
SQL> --Show average income per property in each state
SQL> --Using the income_by_state and properties_per_state views
SQL> SELECT DISTINCT a.state, round(b."State Income"/c."Number",2) AS "Average Income Per
Property"
  2  FROM property a
  3  INNER JOIN income_by_state b
  4  ON a.state = b.state
  5  INNER JOIN properties_per_state c
```

```
  6  ON a.state = c.state
  7  ORDER BY "Average Income Per Property" DESC;

ST Average Income Per Property
-- ---------------------------
MD                14608.33
WA                    7700
KS                      25

SQL>
SQL> --Query 20: For each zipcode that has at least one current tenant with a valid contract, show
the number of tenants in that zipcode.
SQL> SELECT a.zipcode, COUNT(c.fk_tenant_code) AS "Number of Tenants"
  2  FROM property a
  3  INNER JOIN rental_contract c
  4  ON a.property_nickname = c.fk_property_nickname
  5  INNER JOIN primary_tenant b
  6  ON c.fk_tenant_code = b.tenant_code
  7  WHERE b.tenant_code IN
  8  (
  9     SELECT c.fk_tenant_code
 10      FROM rental_contract c
 11      WHERE c.date_end >
 12      (
```

```
13        SELECT sysdate
14        FROM dual
15     )
16  )
17  AND c.current_tenant = 'Y'
18  GROUP BY a.zipcode;

ZIPCO Number of Tenants
----- ----------------
21040              2
66204              4
21014              1

SQL>
SQL> --END
```

9. Database Administration and Monitoring

a. Roles and Responsibilities

Database Administrator:

The database administrator (DBA) is responsible for maintaining the database, as well as ensuring that it remains secure and operates as intended (“What is a Database Administrator (DBA)”, 2022). The DBA will also design any new features, views, etc. that are requested.

System Administrator:

The system administrator will ensure that the computers systems are running smoothly with up-to-date software and security, as well as optimal performance. On the first Tuesday of every month, the systems administrator will run a check on the system to ensure that everything is as it should be.

b. System Information

DBMS: Oracle Database 12c Enterprise Edition, Release 21c 64-bit System requirements ("Oracle Database Documentation", 2022):

- Minimum 4 CPU cores per server
- Minimum 16 GB RAM per server
- 10 GB free hard disk space per server, prior to DBMS installation and database configuration.

c. Performance Monitoring and Database Efficiency

The performance and efficiency of the database will be the responsibility of both the database administrator and the system administrator, with the majority of the responsibility falling on the system administrator. At least once a month, a check will be performed by the system administrator to ensure that the system is performing as it should be.

d. Backup and Recovery

Backup and recovery will be done using Oracle's Recovery Manager (RMAN) (" Oracle Database Backup and Recovery User's Guide", 2015). Initially, a full backup will be done, thereafter (differential) incremental backups will be done twice a day.

10. References

Oracle Database Backup and Recovery User's Guide. (2015) Oracle. https://docs.oracle.com/cd/E11882_01/backup.112/e10642/title.htm

Gregoir, S., Hutin, M., Maury, T., Prandi, G. (2012). Measuring Local Individual Housing Returns from a Large Transaction Database. *Annuls of Economics and Statistics, 107* (108). 93 – 131. DOI 10.2307/23646573a

Nguyen, S. (2022, May 25). *The Benefits of Oracle DBMS for Your Organization*. Dream Factory. https://blog.dreamfactory.com/the-benefits-of-oracle-dbms-for-your-organization/

Oracle Database Documentation. (2022). Oracle. https://docs.oracle.com/en/database/oracle/oracle-database/21/ntdbi/oracle-database-installation-checklist.html

What is a Database Administrator (DBA). (2022) Oracle. https://www.oracle.com/database/what-is-a-dba/

11. Appendix A