University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 2

Date: March 3rd, 2025

Group Number: 80

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Cheuk-lun Cheung	36251726	d2n8s	cheukluncheung@gmail.com
Ali Wagih	21605143	v8i3u	awagih@outlook.com
Ziad Khalifa	22016142	f5f1b	khalifa1@student.ubc.ca

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above.

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

University of British Columbia, Vancouver
Department of Computer Science

1. A completed cover page (template on Canvas)

2. A brief (~2-3 sentences) summary of your project. Many of your TAs are managing multiple projects so this will help them remember details about your project.

Our project is based in the Property Management domain, which focuses on overseeing, maintaining, and managing real estate properties, including residential and commercial units. The database models key aspects such as property details, tenant and ownership management, financial transactions, lease tracking, and maintenance coordination. Users, primarily property managers, can store and track lease agreements, monitor rental income and expenses, submit and follow maintenance requests, and generate reports for owners and stakeholders. We based it on our knowledge of the operations of local firms such as Orca Realty and Bodewell which are known for their efficient management of property portfolios.

3. The ER diagram you are basing your item #3 (below) on. This ER diagram may be the same as your milestone 1 submission or it might be different. If you have made changes from the version submitted in milestone 1, attach a note indicating what changes have been made and why. If you have decided not to implement the suggestions given by your project mentor, please be sure to leave a note stating why. This is not to say that you must do everything that your project mentor says. In many instances, there are trade-offs between design choices and your decision may be influenced by different factors. Your TAs will often leave suggestions that are meant to help massage your project into a form that will fit with the requirements in future project milestones. If you choose not to take their advice, it would be helpful for them to know why to better assist the group moving forward.

The ER diagram below has been slightly modified from Milestone 1 to further improve it.
The changes are as follows:

1. The attribute “propertyMgrContactInfo” was changed to a composite attribute with components “email” and “phoneNumber”.
2. The line underneath the primary key for the Amenity entity, “amenityID”, was changed to a dashed line to reflect that it is a partially unique attribute since Amenity is a weak entity.
3. MaintenanceRequest’s participation in the “makes”, “has”, and “updates” relationships were changed from partial to total. This is because each maintenance request must be made by exactly one tenant, updated by one property manager, and is associated with a single property.
4. The relationship “holds” between “Owner” and “Property” was changed from n:m to 1:n. In other words, each property can be held by exactly one owner, but an owner can have one or more properties.
5. The “propertyID” attribute for “Amenity” isn’t necessary and was instead replaced with “amenitySize”. This is because propertyID is the owner’s primary key and will therefore be inherited by Amenity. There is no need for duplicating it.

University of British Columbia, Vancouver
Department of Computer Science

4. The schema derived from your ER diagram (above). For the translation of the ER diagram to the relational model, follow the same instructions as in your lectures. The process should be reasonably straightforward. For each table:

a. List the table definition (e.g., Table1(attr1: domain1, attr2: domain2, ...)). Make sure to include the domains for each attribute.

b. Specify the primary key (PK), candidate key, (CK) foreign keys (FK), and other constraints (e.g., not null, unique, etc.) that the table must maintain.

```
PropertyManager(
  propertyMgrID: INT NOT NULL,
  propertyMgrName: VARCHAR(50) NOT NULL,
  phoneNumber: VARCHAR(15) NOT NULL,
  email: VARCHAR(50) NOT NULL,
  PRIMARY KEY (propertyMgrID)
)

MaintenanceRequest(
  requestID: INT NOT NULL,
  requestPriority: INT NOT NULL,
  requestStatus: VARCHAR(50) NOT NULL,
  tenantID: INT NOT NULL,
  propertyMgrID: INT NOT NULL,
  propertyID: INT NOT NULL,
  PRIMARY KEY (requestID),
  FOREIGN KEY (tenantID) REFERENCES Tenant(tenantID),
  FOREIGN KEY (propertyMgrID) REFERENCES PropertyManager(propertyMgrID),
  FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
)

Property(
  propertyID: INT NOT NULL,
  propertySize: INT NOT NULL,
  currentValue: FLOAT NOT NULL,
  purchasePrice: FLOAT NOT NULL,
  propertyAddress: VARCHAR(100) NOT NULL,
  propertyMgrID: INT NOT NULL,
  ownerID: INT NOT NULL,
  PRIMARY KEY (propertyID),
  FOREIGN KEY (propertyMgrID) REFERENCES PropertyManager(propertyMgrID),
  FOREIGN KEY (ownerID) REFERENCES Owner(ownerID)
)

Amenity(
  amenityID: INT NOT NULL,
  amentiySize: INT NOT NULL,
  amenityType: VARCHAR(50) NOT NULL,
  propertyID: INT NOT NULL,
```

```
  PRIMARY KEY (amenityID, propertyID),
  FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
  UNIQUE ( amenityType, amenitySize)
)

FinancialTransaction(
  transactionID: INT NOT NULL,
  transactionDate: DATE NOT NULL,
  transactionType: VARCHAR(50) NOT NULL,
  transactionAmount: FLOAT NOT NULL,
  propertyID: INT,
  PRIMARY KEY (transactionID),
  FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
)

Tenant(
  tenantID: INT NOT NULL,
  tenantName: VARCHAR(50) NOT NULL,
  tenantType: VARCHAR(50) NOT NULL,
  propertyID: INT NOT NULL,
  PRIMARY KEY (tenantID),
  FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
)

RentalAgreement(
  contractID: INT NOT NULL,
  monthlyRent: FLOAT NOT NULL,
  securityDeposit: FLOAT NOT NULL,
  tenantID: INT NOT NULL,
  PRIMARY KEY (contractID),
  FOREIGN KEY (tenantID) REFERENCES Tenant(tenantID)
)

Residential(
  resPropertyType: VARCHAR(50) NOT NULL,
  propertyID: INT NOT NULL,
  PRIMARY KEY (propertyID),
  FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
)

Commercial(
  numOfUnits: INT NOT NULL,
  commPropertyType: VARCHAR(50) NOT NULL,
  propertyID: INT NOT NULL,
  PRIMARY KEY (propertyID),
  FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
)

Owner(
  ownerID: INT NOT NULL,
```

```
  ownershipPercentage: FLOAT NOT NULL,
  ownerName: VARCHAR(50) NOT NULL,
  tenantID: INT NOT NULL,
  contractID: INT NOT NULL,
  PRIMARY KEY (ownerID),
  FOREIGN KEY (tenantID) REFERENCES Tenant(tenantID),
  FOREIGN KEY (contractID) REFERENCES RentalAgreement(contractID)
)

Individual(
  socialInsuranceNum: INT NOT NULL,
  creditScore: INT NOT NULL,
  ownerID: INT NOT NULL,
  PRIMARY KEY (ownerID),
  FOREIGN KEY (ownerID) REFERENCES Owner(ownerID)
)

Company(
  businessNum: INT NOT NULL,
  businessType: VARCHAR(50) NOT NULL,
  ownerID: INT NOT NULL,
  PRIMARY KEY (ownerID),
  FOREIGN KEY (ownerID) REFERENCES Owner(ownerID)
)
```

5. Functional Dependencies (FDs)

a. Identify the functional dependencies in your relations, including the ones involving all candidate keys (including the primary key). PKs and CKs are considered functional dependencies and should be included in the list of FDs. You do not need to include trivial FDs such as A -> A. Note: In your list of FDs, there must be some kind of valid FD other than those identified by a PK or CK. If you observe that no relations have FDs other than the PK and CK(s), then you will have to intentionally add some (meaningful) attributes to show valid FDs. We want you to get a good normalization exercise. Your design must go through a normalization process. You do not need to have a non-PK/CK FD for each relation but be reasonable. If your TA feels that some non-PK/CK FDs have been omitted, your grade will be adjusted accordingly.

PropertyManager

1. propertyMgrID -> propertyMgrName, phoneNumber, email
2. phoneNumber -> propertyMgrID

MaintenanceRequest

1. requestID -> requestPriority, requestStatus, tenantID, propertyMgrID, propertyID
2. requestPriority -> requestStatus

Property

1. propertyID -> propertySIze, currentValue, purchasePrice, propertyAddress, propertyMgrID, ownerID

2. propertyAddress -> propertyID

Amenity
1. (amenityID, propertyID) -> amenityType, amenitySize
2. (propertyID, amenityType, amenitySize) -> amenityID

FinancialTransactions
1. transactionID -> transactionDate, transactionType, transactionAmount, propertyID

Tenant
1. tenantID -> tenantName, tenantType, propertyID
2. (tenantName, propertyID) -> tenantID

RentalAgreement
1. contractID -> monthlyRent, securityDeposit, tenantID
2. tenantID -> contractID

Residential
1. propertyID -> resPropertyType

Commercial
1. propertyID -> numOfUnits, commPropertyType

Owner
1. ownerID -> ownershipPercentage, ownerName, tenantID, contractID

Individual
1. ownerID -> socialInsuranceNum, creditScore
2. socialInsuranceNum -> ownerID

Company
1. ownerID -> businessNum, businessType
2. businessNum -> ownerID

6. Normalization

a. Normalize each of your tables to be in 3NF or BCNF. Give the list of tables, their primary keys, their candidate keys, and their foreign keys after normalization. You should show the steps taken for the decomposition in a manner similar to that done in class. Should there be errors, and no work is shown, no partial credit can be awarded without steps shown. The format should be the same as Step 3, with tables listed similar to Table1(attr1:domain1, attr2:domain2, …). ALL Tables must be listed, not only the ones post normalization.

University of British Columbia, Vancouver
Department of Computer Science

1. PropertyManager

Original Definition:

```
PropertyManager(
    propertyMgrID: INT NOT NULL,
    propertyMgrName: VARCHAR(50) NOT NULL,
    phoneNumber: VARCHAR(15) NOT NULL,
    email: VARCHAR(50) NOT NULL,
    PRIMARY KEY (propertyMgrID)
)
```

Functional Dependencies:

- propertyMgrID → propertyMgrName, phoneNumber, email
- phoneNumber → propertyMgrID

Candidate Keys:

– propertyMgrID
– phoneNumber

Foreign Keys:

– None

Normalization Status:
Already in BCNF (all determinants are candidate keys).

2. MaintenanceRequest

Original Definition:

```
MaintenanceRequest(
    requestID: INT NOT NULL,
    requestPriority: INT NOT NULL,
    requestStatus: VARCHAR(50) NOT NULL,
    tenantID: INT NOT NULL,
    propertyMgrID: INT NOT NULL,
    propertyID: INT NOT NULL,
    PRIMARY KEY (requestID),
    FOREIGN KEY (tenantID) REFERENCES Tenant(tenantID),
    FOREIGN KEY (propertyMgrID) REFERENCES PropertyManager(propertyMgrID),
    FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
)
```

Functional Dependencies:

- requestID → requestPriority, requestStatus, tenantID, propertyMgrID, propertyID
- requestPriority → requestStatus

The FD requestPriority → requestStatus violates BCNF (and 3NF) because requestPriority is not a candidate key (the only candidate key here is requestID).

Decomposition Steps:

1. Remove the dependency requestPriority → requestStatus by splitting the relation.
2. Create one relation to hold the "core" information (key and FDs that depend on it) and a second relation to map each requestPriority to its unique requestStatus.

Decomposed Relations:

```
MaintenanceRequest(
    requestID: INT NOT NULL,
    requestPriority: INT NOT NULL,
    tenantID: INT NOT NULL,
    propertyMgrID: INT NOT NULL,
    propertyID: INT NOT NULL,
    PRIMARY KEY (requestID),
    FOREIGN KEY (tenantID) REFERENCES Tenant(tenantID),
    FOREIGN KEY (propertyMgrID) REFERENCES PropertyManager(propertyMgrID),
    FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
)
```

Candidate Key: requestID

```
MaintenanceRequestStatus(
    requestPriority: INT NOT NULL,
    requestStatus: VARCHAR(50) NOT NULL,
    PRIMARY KEY (requestPriority)
)
```

Candidate Key: requestPriority

After decomposition, both tables are in BCNF.

3. Property

Original Definition:

```
Property(
    propertyID: INT NOT NULL,
    propertySize: INT NOT NULL,
    currentValue: FLOAT NOT NULL,
    purchasePrice: FLOAT NOT NULL,
    propertyAddress: VARCHAR(100) NOT NULL,
    propertyMgrID: INT NOT NULL,
    ownerID: INT NOT NULL,
    PRIMARY KEY (propertyID),
    FOREIGN KEY (propertyMgrID) REFERENCES PropertyManager(propertyMgrID),
    FOREIGN KEY (ownerID) REFERENCES Owner(ownerID)
)
```

Functional Dependencies:

- propertyID → propertySize, currentValue, purchasePrice, propertyAddress, propertyMgrID, ownerID
- propertyAddress → propertyID

Candidate Keys:
- propertyID
- propertyAddress

Foreign Keys:
- propertyMgrID → PropertyManager
- ownerID → Owner

Normalization Status:
Already in BCNF.

4. Amenity

Original Definition:

```
Amenity(
  amenityID: INT NOT NULL,
  amentiySize: INT NOT NULL,
  amenityType: VARCHAR(50) NOT NULL,
  propertyID: INT NOT NULL,
  PRIMARY KEY (amenityID, propertyID),
  FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
  UNIQUE ( amenityType, amenitySize)
)
```

Functional Dependencies:

- (amenityID, propertyID) -> amenityType, amenitySize
- (propertyID, amenityType, amenitySize) -> amenityID

The FD (propertyID, amenityType, amenitySize) -> amenityID violates BCNF because LHS (propertyID, amenityType, amenitySize) is not a Superkey.

Decomposition Steps:

1. Split the Amenity relation into two relations, one for the AmenityType and one for the Amenity.

Decomposed Relations:

```
AmenityType(
propertyID INT NOT NULL,
amenityType INT NOT NULL,
amenitySize INT NOT NULL,
PRIMARY KEY (propertyID, amenityType, amenitySize)
)
```

Candidate Key: propertyID, amenityType, amenitySize

Amenity(
amenityID INT NOT NULL,
propertyID INT NOT NULL,
amenityType INT NOT NULL,
amenitySize INT NOT NULL,
PRIMARY KEY (amenityID),
FOREIGN KEY (propertyID, amenityType, amenitySize) REFERENCES AmenityType(propertyID, amenityType, amenitySize)
)

Candidate Key: amenityID

After decomposition, both tables are in BCNF.

5. FinancialTransaction

Original Definition:

```
FinancialTransaction(
    transactionID: INT NOT NULL,
    transactionDate: DATE NOT NULL,
    transactionType: VARCHAR(50) NOT NULL,
    transactionAmount: FLOAT NOT NULL,
    propertyID: INT,
    PRIMARY KEY (transactionID),
    FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
)
```

Functional Dependencies:

- transactionID → transactionDate, transactionType, transactionAmount, propertyID

Candidate Keys:

– transactionID

Foreign Keys:

– propertyID → Property

Normalization Status:
Already in BCNF.

6. Tenant

Original Definition:

```
Tenant(
    tenantID: INT NOT NULL,
    tenantName: VARCHAR(50) NOT NULL,
    tenantType: VARCHAR(50) NOT NULL,
    propertyID: INT NOT NULL,
    PRIMARY KEY (tenantID),
    FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
)
```

Functional Dependencies:
- tenantID → tenantName, tenantType, propertyID
- (tenantName, propertyID) → tenantID

Candidate Keys:
- tenantID
- (tenantName, propertyID)

Foreign Keys:
- propertyID → Property

Normalization Status:
Already in BCNF.

7. RentalAgreement

Original Definition:

```
RentalAgreement(
    contractID: INT NOT NULL,
    monthlyRent: FLOAT NOT NULL,
    securityDeposit: FLOAT NOT NULL,
    tenantID: INT NOT NULL,
    PRIMARY KEY (contractID),
    FOREIGN KEY (tenantID) REFERENCES Tenant(tenantID)
)
```

Functional Dependencies:
- contractID → monthlyRent, securityDeposit, tenantID
- tenantID → contractID

Candidate Keys:
- contractID
- tenantID

Foreign Keys:
- tenantID → Tenant

Normalization Status:
Already in BCNF.

8. Residential

Original Definition:

```
Residential(
    resPropertyType: VARCHAR(50) NOT NULL,
    propertyID: INT NOT NULL,
    PRIMARY KEY (propertyID),
    FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
)
```

Functional Dependencies:
- propertyID → resPropertyType

Candidate Keys:
– propertyID

Foreign Keys:
– propertyID → Property

Normalization Status:
Already in BCNF.

9. Commercial

Original Definition:

```
Commercial(
    numOfUnits: INT NOT NULL,
    commPropertyType: VARCHAR(50) NOT NULL,
    propertyID: INT NOT NULL,
    PRIMARY KEY (propertyID),
    FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
)
```

Functional Dependencies:
- propertyID → numOfUnits, commPropertyType

Candidate Keys:
– propertyID

Foreign Keys:
– propertyID → Property

Normalization Status:
Already in BCNF.

10. Owner

Original Definition:

```
Owner(
    ownerID: INT NOT NULL,
    ownershipPercentage: FLOAT NOT NULL,
    ownerName: VARCHAR(50) NOT NULL,
    tenantID: INT NOT NULL,
    contractID: INT NOT NULL,
    PRIMARY KEY (ownerID),
    FOREIGN KEY (tenantID) REFERENCES Tenant(tenantID),
    FOREIGN KEY (contractID) REFERENCES RentalAgreement(contractID)
)
```

Functional Dependencies:
- ownerID → ownershipPercentage, ownerName, tenantID, contractID

Candidate Keys:
- ownerID

Foreign Keys:
- tenantID → Tenant
- contractID → RentalAgreement

Normalization Status:
Already in BCNF.

11. Individual

Original Definition:

```
Individual(
    socialInsuranceNum: INT NOT NULL,
    creditScore: INT NOT NULL,
    ownerID: INT NOT NULL,
    PRIMARY KEY (ownerID),
    FOREIGN KEY (ownerID) REFERENCES Owner(ownerID)
)
```

Functional Dependencies:
- ownerID → socialInsuranceNum, creditScore
- socialInsuranceNum → ownerID

Candidate Keys:
- ownerID
- socialInsuranceNum

Foreign Keys:
- ownerID → Owner

Normalization Status:
Already in BCNF.

12. Company

Original Definition:

```
Company(
    businessNum: INT NOT NULL,
    businessType: VARCHAR(50) NOT NULL,
    ownerID: INT NOT NULL,
    PRIMARY KEY (ownerID),
    FOREIGN KEY (ownerID) REFERENCES Owner(ownerID)
)
```

Functional Dependencies:
- ownerID → businessNum, businessType
- businessNum → ownerID

Candidate Keys:
- ownerID
- businessNum

Foreign Keys:
- ownerID → Owner

Normalization Status:
Already in BCNF.

7. The SQL DDL statements required to create all the tables from item #6. The statements should use the appropriate foreign keys, primary keys, UNIQUE constraints, etc. Unless you know that you will always have exactly x characters for a given character, it is better to use the VARCHAR data type as opposed to a CHAR(Y). For example, UBC courses always use four characters to represent which department offers a course. In that case, you will want to use CHAR(4) for the department attribute in your SQL DDL statement. If you are trying to represent the name of a UBC course, you will want to use VARCHAR as the number of characters in a course name can vary greatly.

```
CREATE TABLE PropertyManager
(
 propertyMgrID INT NOT NULL,
 propertyMgrName VARCHAR(50) NOT NULL,
 phoneNumber VARCHAR(15) NOT NULL,
 email VARCHAR(50) NOT NULL,
 PRIMARY KEY (propertyMgrID)
);

CREATE TABLE MaintenanceRequestStatus
(
 requestPriority INT NOT NULL,
 requestStatus VARCHAR(50) NOT NULL,
 PRIMARY KEY (requestPriority)
);

CREATE TABLE MaintenanceRequest
(
 requestID INT NOT NULL,
 requestPriority INT NOT NULL,
 tenantID INT NOT NULL,
 propertyMgrID INT NOT NULL,
 propertyID INT NOT NULL,
 PRIMARY KEY (requestID),
 FOREIGN KEY (tenantID) REFERENCES Tenant(tenantID),
```

```
  FOREIGN KEY (propertyMgrID) REFERENCES PropertyManager(propertyMgrID),
  FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
);

CREATE TABLE Property
(
  propertyID INT NOT NULL,
  propertySize INT NOT NULL,
  currentValue FLOAT NOT NULL,
  purchasePrice FLOAT NOT NULL,
  propertyAddress VARCHAR(100) NOT NULL,
  propertyMgrID INT NOT NULL,
  ownerID INT NOT NULL,
  PRIMARY KEY (propertyID),
  FOREIGN KEY (propertyMgrID) REFERENCES PropertyManager(propertyMgrID),
  FOREIGN KEY (ownerID) REFERENCES Owner(ownerID)
);

CREATE TABLE AmenityType
(
  propertyID INT NOT NULL,
  amenityType INT NOT NULL,
  amenitySize INT NOT NULL,
  PRIMARY KEY (propertyID, amenityType, amenitySize)
);


CREATE TABLE Amenity
(
  amenityID INT NOT NULL,
  propertyID INT NOT NULL,
  amenityType INT NOT NULL,
  amenitySize INT NOT NULL,
  PRIMARY KEY (amenityID),
  FOREIGN KEY (propertyID, amenityType, amenitySize) REFERENCES AmenityType(propertyID,
amenityType, amenitySize)
);


CREATE TABLE FinancialTransaction
(
  transactionID INT NOT NULL,
  transactionDate DATE NOT NULL,
  transactionType VARCHAR(50) NOT NULL,
  transactionAmount FLOAT NOT NULL,
  propertyID INT,
  PRIMARY KEY (transactionID),
  FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
);
```

```
CREATE TABLE Tenant
(
 tenantID INT NOT NULL,
 tenantName VARCHAR(50) NOT NULL,
 tenantType VARCHAR(50) NOT NULL,
 propertyID INT NOT NULL,
 PRIMARY KEY (tenantID),
 FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
);

CREATE TABLE RentalAgreement
(
 contractID INT NOT NULL,
 monthlyRent FLOAT NOT NULL,
 securityDeposit FLOAT NOT NULL,
 tenantID INT NOT NULL,
 PRIMARY KEY (contractID),
 FOREIGN KEY (tenantID) REFERENCES Tenant(tenantID)
);

CREATE TABLE Residential
(
 resPropertyType VARCHAR(50) NOT NULL,
 propertyID INT NOT NULL,
 PRIMARY KEY (propertyID),
 FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
);

CREATE TABLE Commercial
(
 numOfUnits INT NOT NULL,
 commPropertyType VARCHAR(50) NOT NULL,
 propertyID INT NOT NULL,
 PRIMARY KEY (propertyID),
 FOREIGN KEY (propertyID) REFERENCES Property(propertyID)
);

CREATE TABLE Owner
(
 ownerID INT NOT NULL,
 ownershipPercentage FLOAT NOT NULL,
 ownerName VARCHAR(50) NOT NULL,
 tenantID INT NOT NULL,
 contractID INT NOT NULL,
 PRIMARY KEY (ownerID),
 FOREIGN KEY (tenantID) REFERENCES Tenant(tenantID),
 FOREIGN KEY (contractID) REFERENCES RentalAgreement(contractID)
);

CREATE TABLE Individual
```

```
(
  socialInsuranceNum INT NOT NULL,
  creditScore INT NOT NULL,
  ownerID INT NOT NULL,
  PRIMARY KEY (ownerID),
  FOREIGN KEY (ownerID) REFERENCES Owner(ownerID)
);

CREATE TABLE Company
(
  businessNum INT NOT NULL,
  businessType VARCHAR(50) NOT NULL,
  ownerID INT NOT NULL,
  PRIMARY KEY (ownerID),
  FOREIGN KEY (ownerID) REFERENCES Owner(ownerID)
);
```

8. INSERT statements to populate each table with at least 5 tuples. You will likely want to have more than 5 tuples so that you can have meaningful queries later.

```
INSERT INTO PropertyManager (propertyMgrID, propertyMgrName, phoneNumber, email) VALUES
  (1, 'Alice Manager', '555-1111', 'alice.manager@example.com'),
  (2, 'Bob Manager', '555-2222', 'bob.manager@example.com'),
  (3, 'Carol Manager', '555-3333', 'carol.manager@example.com'),
  (4, 'Dave Manager', '555-4444', 'dave.manager@example.com'),
  (5, 'Eve Manager', '555-5555', 'eve.manager@example.com');

INSERT INTO Owner (ownerID, ownershipPercentage, ownerName, tenantID, contractID) VALUES
  (1, 100.0, 'John Owner', 1, 1),
  (2,  50.0, 'Mary Owner', 2, 2),
  (3,  75.0, 'Steve Owner', 3, 3),
  (4,  60.0, 'Lucy Owner', 4, 4),
  (5,  80.0, 'Mark Owner', 5, 5),
  (11, 100.0, 'Alpha Corp', 1, 1),
  (12,  80.0, 'Beta Corp', 2, 2),
  (13,  90.0, 'Charlie Inc', 3, 3),
  (14,  70.0, 'Delta Ltd', 4, 4),
  (15,  85.0, 'Epsilon LLC', 5, 5);

INSERT INTO Property (propertyID, propertySize, currentValue, purchasePrice, propertyAddress,
propertyMgrID, ownerID) VALUES
  (1, 1500, 300000.0, 250000.0, '123 Maple St', 1, 1),
  (2, 2000, 450000.0, 400000.0, '456 Oak Ave', 2, 2),
  (3, 1800, 350000.0, 300000.0, '789 Pine Rd', 3, 3),
  (4, 2200, 500000.0, 450000.0, '101 Cedar Blvd', 4, 4),
  (5, 1600, 320000.0, 280000.0, '202 Birch Ln', 5, 5),
  (6, 1400, 280000.0, 230000.0, '303 Spruce St', 1, 1),
  (7, 1300, 260000.0, 210000.0, '404 Elm St', 2, 2),
  (8, 2500, 600000.0, 550000.0, '505 Walnut Ave', 3, 3),
```

```
  (9, 2600, 620000.0, 570000.0, '606 Cherry St', 4, 4),
  (10,2400, 580000.0, 530000.0, '707 Poplar Rd', 5, 5);

INSERT INTO Tenant (tenantID, tenantName, tenantType, propertyID) VALUES
  (1, 'Tenant One', 'Residential', 1),
  (2, 'Tenant Two', 'Commercial', 2),
  (3, 'Tenant Three', 'Residential', 3),
  (4, 'Tenant Four', 'Commercial', 4),
  (5, 'Tenant Five', 'Residential', 5);

INSERT INTO RentalAgreement (contractID, monthlyRent, securityDeposit, tenantID) VALUES
  (1, 1500.0, 1500.0, 1),
  (2, 2000.0, 2000.0, 2),
  (3, 1800.0, 1800.0, 3),
  (4, 2200.0, 2200.0, 4),
  (5, 1600.0, 1600.0, 5);

INSERT INTO MaintenanceRequestStatus (requestPriority, requestStatus) VALUES
  (1, 'Open'),
  (2, 'Closed'),
  (1, 'In Progress'),
  (3, 'Open'),
  ( 2, 'Closed');

INSERT INTO MaintenanceRequest (requestID, requestPriority, tenantID, propertyMgrID, propertyID)
VALUES
  (1, 1, 1, 1, 1),
  (2, 2, 2, 2, 2),
  (3, 1, 3, 3, 3),
  (4, 3, 4, 4, 4),
  (5, 2, 5, 5, 5);


INSERT INTO AmenityType (amenitysize, amenityType, propertyID) VALUES
  (500, 'Pool',     1),
  (1200, 'Gym',      2),
  (3000, 'Parking',  3),
  (430, 'Garden',   4),
  (50, 'Elevator', 5);

INSERT INTO Amenity (amenityID, amenityType, amenitySize, propertyID) VALUES
  (1, 'Pool', 500, 1),
  (2, 'Gym', 1200, 2),
  (3, 'Parking', 3000, 3),
  (4, 'Garden', 430, 4),
  (5, 'Elevator', 50, 5);

INSERT INTO Residential (propertyID, resPropertyType) VALUES
  (1, 'Apartment'),
  (3, 'Condo'),
```

```
  (5, 'House'),
  (6, 'Townhouse'),
  (7, 'Loft');

INSERT INTO Commercial (propertyID, numOfUnits, commPropertyType) VALUES
  (2, 10, 'Office'),
  (4, 15, 'Retail'),
  (8,  8, 'Office'),
  (9, 12, 'Retail'),
  (10, 9, 'Warehouse');

INSERT INTO Individual (ownerID, socialInsuranceNum, creditScore) VALUES
  (1, 123456789, 750),
  (2, 987654321, 720),
  (3, 192837465, 680),
  (4, 564738291, 700),
  (5, 102938475, 710);

INSERT INTO Company (ownerID, businessNum, businessType) VALUES
  (11, 111111, 'Real Estate'),
  (12, 222222, 'Property Investment'),
  (13, 333333, 'Realty'),
  (14, 444444, 'Asset Management'),
  (15, 555555, 'Property Services');
```

9. An explicit acknowledgment about your use of AI tools in this assignment. Specifically, we are looking for a clear yes/no about whether you have used one or more AI tools. If yes, we want to know which tool(s) you have used and what prompt(s) you have given the tool.

No AI tools were used in this assignment.