Photo/Video Rental DBMS

CPS 510 | Toronto Metropolitan University

Ajmain Hyder

Aanish Mufti

Naureen Hossain

Table of Contents

Introduction

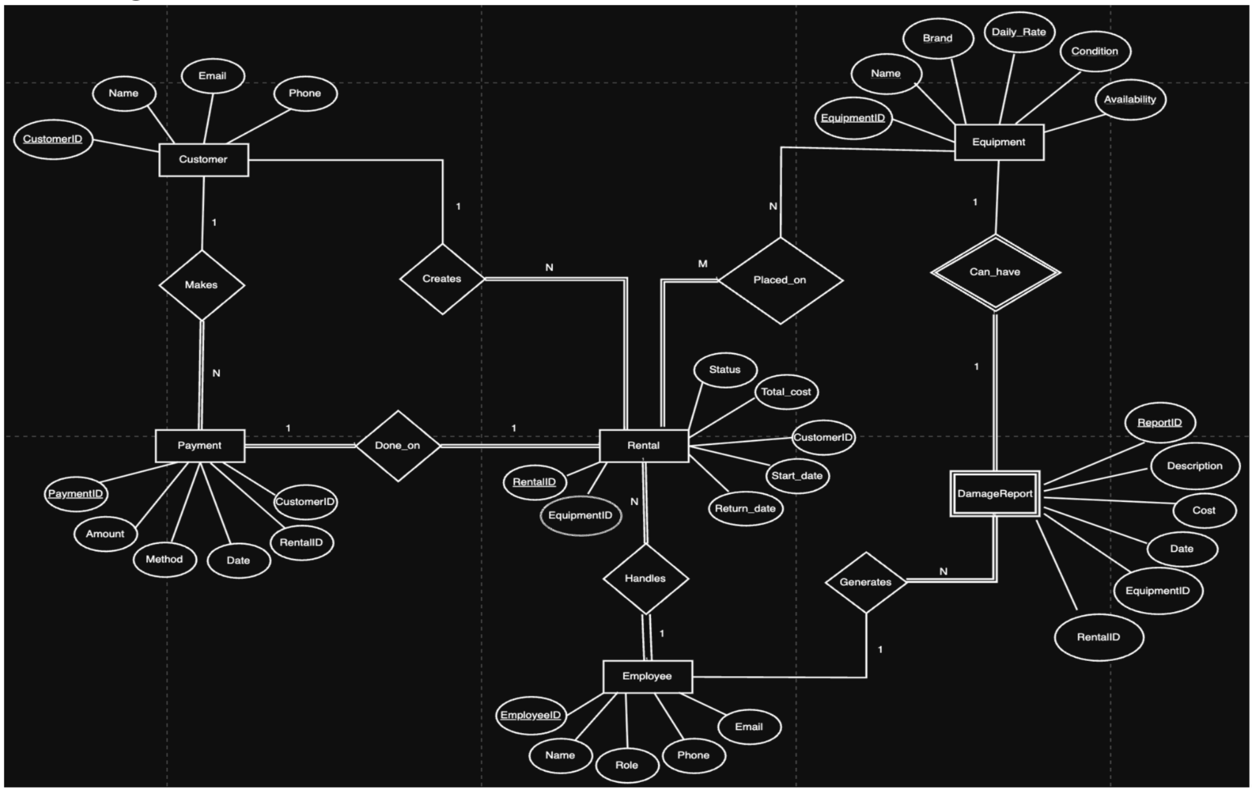
In today's society where media equipment is an essential part for social content, the demand for accessible and reliable services have grown increasingly. The media industry grew 5.5% just in 2024 to 2.9 trillion and is projected to reach 3.5 trillion by 2029 (Spiegel, 2025). Many podcasters and media outlets starting off can't afford a high up front cost to buy the equipment. With many of them renting out these equipment, it is important that rental services have a reliable and accessible management system to keep track of their equipment.

As a result, companies who run their own media rental service need to be organized. Each month, they handle hundreds of clients, pieces of equipment and rental transactions. It becomes challenging to keep track of who rented what, return dates, conditions and whether payments were received. All these records are important and must be kept in one location and accessible with ease, which will require a database management system. This keeps consumers satisfied, simplifies the leasing process, avoids error and ultimately helps the business grow while keeping customers happy.

Function and Application

In order to successfully and efficiently run a rental business the Photo/Video Equipment Rental DBMS would have to support the following functions: inventory/rental management, customer registration and profile management, equipment availability status, payment and billing, and maintenance records. The inventory/rental management function allows the system to keep track of all equipment as well as their types, condition, and rental status. The customer registration and profile management function would allow the system to identify and manage the customers by storing important information such as their name, identification number, contact information, and rental history. The availability check function is essential in order to avoid scheduling conflicts in renting out equipment as it allows both the staff and the customers to check which equipment is available and for how long. If an equipment is rented out, it shows the expected date of return, assuming the customer will return it on time. The payment and billing function handles the calculation of the equipment rentals, the application of late fees or penalties, and the tracking of payments made by the customer in cash, by card, or online. Finally, the maintenance records plays an essential part in ensuring that all the equipment is in perfect condition as it allows the system to monitor and record any damages and the status of the repairs.

ER Diagram



Entities and Attributes

Customer

customer_id	name	email	phone
00001	John Doe	johndoe@gmail.com	111-111-1111
00002	Jane Doe	janedoe@gmail.com	222-222-2222

customer_id is a unique 5 digit number. The Database will store the data of the customers who represent individuals or organizations renting out the equipment.

Equipment

equipment_id	name	type	brand	daily_rate	cost	condition	availability
1000	EOS R8	Photo	Canon	$50	$2,200.0	New	available

1001	FX2	Video	Sony	$500	$4,248.0	Used	unavailable

equipment_id is a unique 4 digit number. Equipment will store the devices that are available for rent. It stores the type of camera, its purchase cost and the daily rate that will be used to charge customers a fee.

Rental

rental_id	Total_Cost	start_date	return_date	status	customer_id	equipment_id
00000001	$300	01/01/25	01/04/25	Due	00001	1000

rental_id is a unique 8 digit number. This entity will store the data relating to the actual renting process. They include information of the equipment, customer, and the duration of the rent itself.

Payment

payment_id	amount	payment_method	date	rental_id	customer_id
0001	$200	Visa	01/01/2025	00000001	00001

Payment_id is a unique 4 digit number. Stores the financial transactions made for the rentals.

Employee

employee_id	Name	Role	Phone	Email
11000	Alen LastName	Sales	+437000000	Alen@email.com

Employee_id is a unique 5 digit number that starts with “11”, example “11_ _ _”. Stores the data of the staff working with the equipment and the store itself.

Damage Report (Weak entity of Equipment)

report_id	description	Cost	date	equipment_id	rental_id
1001	“Broken lens”	$200	01/01/2025	1000	00000001

Report_id is a unique combination of a 3 digit number and the associated normalised rentalID. Example: Rental_id: 00000001 report_id: 1001. Reports created on damaged equipment will also be stored.

Relationships

Customers are able to place multiple rentals, with each rental being associated with a single customer. A rental may include one or more equipment items, while each piece of equipment can appear in multiple rentals over time. Every rental is managed by an employee, and a single employee may be responsible for handling multiple rental transactions. Once a rental has been paid for, a corresponding payment record is created, linking the transaction to both the rental and the customer involved, thereby ensuring accurate financial tracking. In addition, employees may generate damage reports for equipment when necessary, which establishes the dependency of the Damage Report entity on the Equipment entity.

Functional Dependencies

1. Customer

Schema: Customer(CustomerID, Name, Email, Phone)

Functional Dependencies (FDs):

{ CustomerID -> Name, Email, Phone } { Email -> CustomerID, Name, Phone }

Explanation: Each customer is uniquely identified by CustomerID. The Email attribute is also unique, which allows it to determine all other customer details.

2. Equipment

Schema: Equipment(EquipmentID, Name, Brand, Daily_Rate, Condition, Availability)

Functional Dependencies (FDs):

{ EquipmentID -> Name, Brand, Daily_Rate, Condition, Availability }

Explanation: Each piece of equipment is uniquely identified by EquipmentID, which determines all other non-key attributes describing the equipment.

3. Employee

Schema: Employee(EmployeeID, Name, Role, Phone, Email)

Functional Dependencies (FDs):

{ EmployeeID -> Name, Role, Phone, Email } { Email -> EmployeeID, Name, Role, Phone }

Explanation: Each employee is uniquely identified by EmployeeID. The Email field is also unique, providing an alternate key that determines all other employee attributes.

4. Rental

Schema: Rental(RentalID, CustomerID, EquipmentID, Status, Total_Cost, Start_Date, Return_Date)

Functional Dependencies (FDs):

{ RentalID -> CustomerID, EquipmentID, Status, Total_Cost, Start_Date, Return_Date } { CustomerID, EquipmentID, Start_Date -> RentalID, Status, Total_Cost, Return_Date }

Explanation: Each rental transaction has a unique RentalID. Additionally, a combination of CustomerID, EquipmentID, and Start_Date can also uniquely determine a rental record.

5. Payment

Schema: Payment(PaymentID, RentalID, CustomerID, Amount, MethodUsed, PaymentDate)

Functional Dependencies (FDs):

{ PaymentID -> RentalID, CustomerID, Amount, MethodUsed, PaymentDate } { RentalID -> CustomerID }

Explanation: Each payment is uniquely identified by PaymentID. Since each rental is made by a single customer, RentalID determines CustomerID.

6. DamageReport

Schema: DamageReport(ReportID, RentalID, EquipmentID, Description, Cost, ReportDate)

Functional Dependencies (FDs):

{ ReportID -> RentalID, EquipmentID, Description, Cost, ReportDate } { RentalID, EquipmentID -> Description, Cost, ReportDate }

Explanation: Each damage report is identified by a unique ReportID. Additionally, a specific combination of RentalID and EquipmentID can determine the report details for that rental.

7. Handles

Schema: Handles(EmployeeID, RentalID)

Functional Dependencies (FDs): { EmployeeID, RentalID -> — }

Explanation: This is a relationship (junction) table linking Employee and Rental. It contains only key attributes and no non-key attributes, so there are no additional dependencies.

Normalization/3NF/BCNF

3NF Table Requirement:

- Must be in 1NF: All values are atomic (no groups or lists are repeated)
- Must be in 2NF: If it is in 1NF and every non-key attribute is fully functionally dependent on the whole primary key.
- Every non-key attribute must be nontransitively dependent on the primary key.

Algorithm for 3NF:

Step 1: Identify all functional dependencies .
Step 2: Identify all candidate keys for each relation.
Step 3: For each functional dependency X -> Y:

- If X is a superkey, keep it in the same relation.
- If Y contains only prime attributes, it can stay
- Otherwise, decompose the relation into smaller ones so each non-key attribute depends only on a key

Verifying that all Tables are in 3NF:

Table	Primary Key	1NF	2NF	3NF	Explanation
Customer	CustomerID	Yes	Yes	Yes	All attributes (Name, Email, Phone, Address) depend solely on CustomerID. Each field is atomic (one email, one phone, etc.). Email is unique but not a transitive dependency, since it directly identifies the customer.
Equipment	EquipmentID	Yes	Yes	Yes	EquipmentID uniquely identifies each item. Attributes like Name, Brand, Rate, Condition, and Availability depend directly on it. There are no repeating groups or

					derived attributes.
Employee	EmployeeID	Yes	Yes	Yes	Attributes (Name, Role, Phone, Email) depend only on the employee’s ID. There are no dependencies among non-key attributes, so no transitive dependency exists.
Rental	RentalID	Yes	Yes	Yes	Each rental is uniquely identified by RentalID. Other fields (CustomerID, EquipmentID, StartDate, EndDate, TotalCost) depend on this key. CustomerID and EquipmentID are foreign keys, but not determinants of other attributes in this table.
Payment	PaymentID	Yes	Yes	Yes	All non-key attributes (Amount, PaymentDate, PaymentType, RentalID, CustomerID) depend directly on PaymentID. The foreign keys reference other entities but do not introduce transitive dependencies.
DamageReport	ReportID	Yes	Yes	Yes	ReportID uniquely identifies each record. Description, RepairCost, and DateReported depend only on this key. RentalID and EquipmentID are foreign keys with no partial or transitive dependency.
Handles	EmployeeID RentalID	Yes	Yes	Yes	Composite key formed by both columns. The relationship indicates which employee handled which rental. No other attributes depend on part of the key, so it meets 2NF. There are also no transitive dependencies.

As all of our tables where already on 3NF, an example of table decomposition is given below:

Example Table: CustomerRental

Schema:

CustomerRental (CustomerID, EquipmentID, CustomerName, EquipmentName, DailyRate, RentalDays, TotalCost)

Functional Dependencies:

CustomerID -> CustomerName
EquipmentID -> EquipmentName, DailyRate
CustomerID, EquipmentI -> RentalDays, TotalCost
EquipmentID, RentalDays -> TotalCost

Candidate Keys:

{CustomerID, EquipmentID}

Finally we can decompose the The table into 3 different ones:

1. Customer(CustomerID, CustomerName)
   FD: CustomerID -> CustomerName
2. Equipment(EquipmentID, EquipmentName, DailyRate)
   FD: EquipmentID -> EquipmentName, DailyRate
3. Rental (CustomerID, EquipmentID, RentalDays)
   FD: CustomerID, EquipmentID -> RentalDays

BCNF Verification

Customer:

FD: CustomerID -> Name, Email, Phone
Email → CustomerID, Name, Phone

Candidate Keys: {CustomerID}, {Email}

Both FDs have determinants (CustomerID and Email) that are **candidate keys**.

Equipment:

FD: EquipmentID -> Name, Brand, Daily_Rate, Condition, Availability

Candidate Key: {EquipmentID}

EquipmentID is a key, so this table satisfies BCNF.

Employee:

FD: EmployeeID -> Name, Role, Phone, Email

Candidate Keys: {EmployeeID}, {Email}
Both EmployeeID and Email are keys, therefore they are in BCNF

Rental:

FD: RentalID -> CustomerID, EquipmentID, Status, Total_Cost, Start_Date,Return_Date

Candidate Key: {RentalID}
RentalID is a key, so this table satisfies BCNF.

Payment

FD: PaymentID -> RentalID, CustomerID, Amount, MethodUsed, PaymentDate

Candidate Keys: {PaymentID},
PaymentID is a key, so this table satisfies BCNF

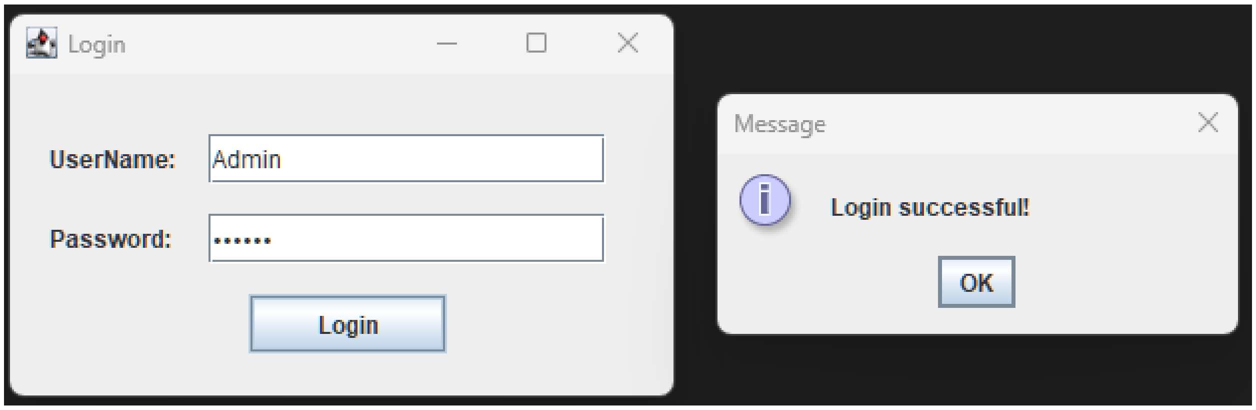
DamageReport

FD: ReportID -> RentalID, EquipmentID, Description, Cost, ReportDate

Candidate Key: {ReportID}
ReportID is a key, so this table satisfies BCNF.

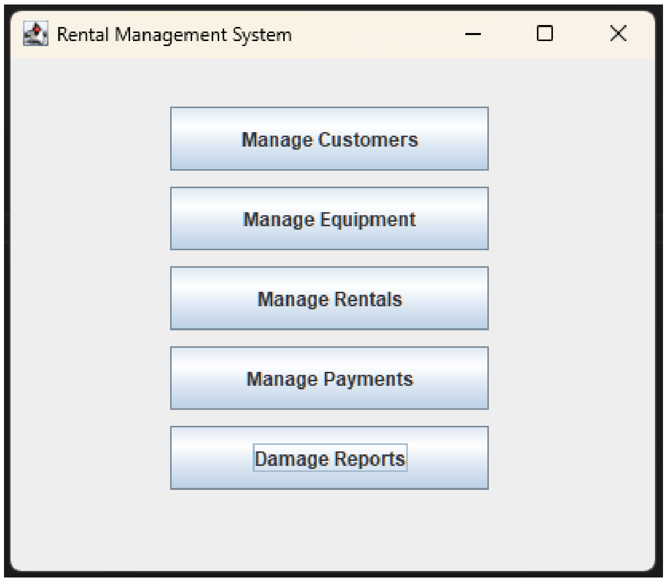
Java UI app

Screenshots Demonstrating the Apps Functions:

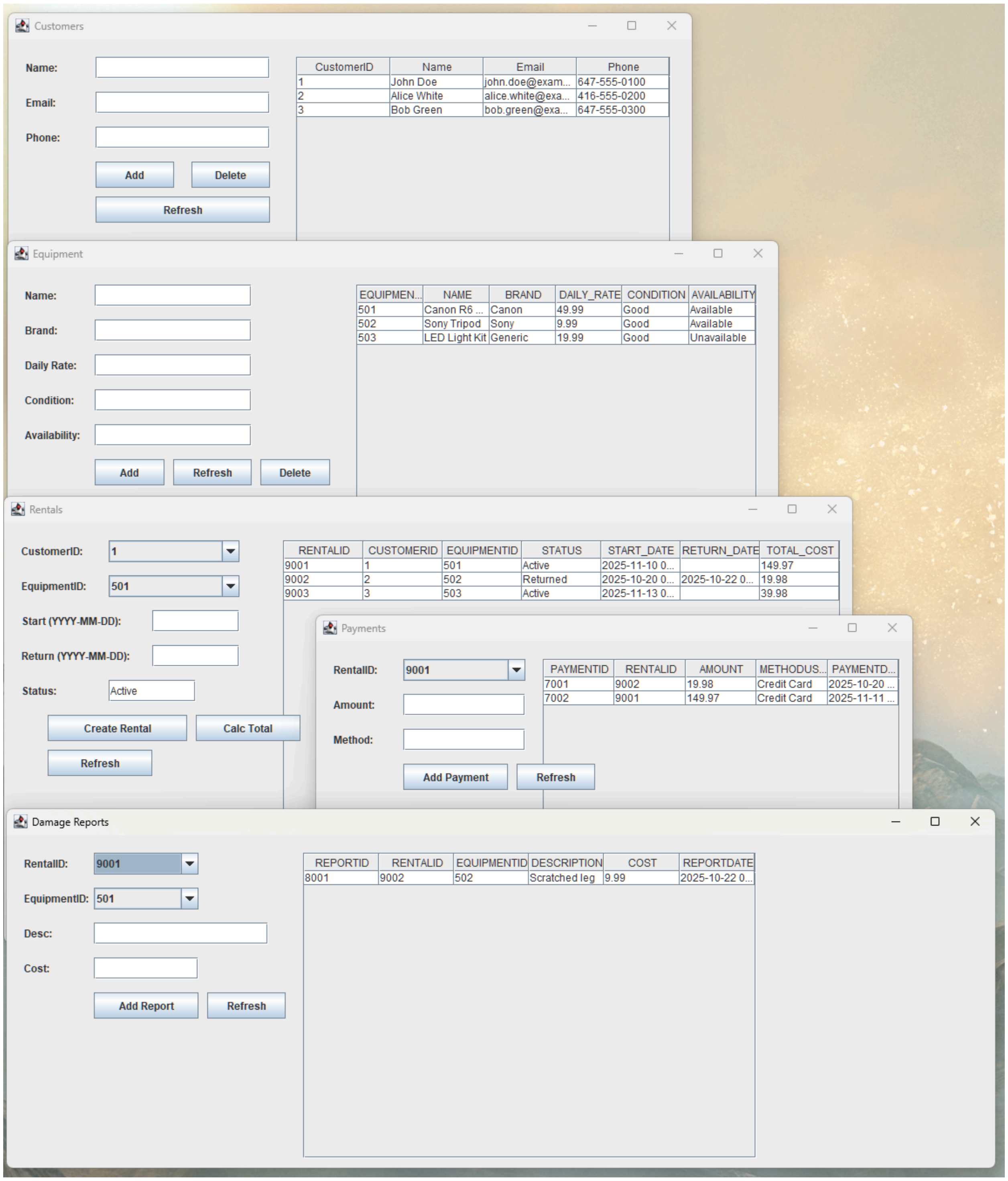


Login successful!

- Login Screen



- UI of Main Screen



CustomerID	Name	Email	Phone
1	John Doe	john.doe@exam...	647-555-0100
2	Alice White	alice.white@exa...	416-555-0200
3	Bob Green	bob.green@exa...	647-555-0300

EQUIPMEN...	NAME	BRAND	DAILY_RATE	CONDITION	AVAILABILITY
501	Canon R6 ...	Canon	49.99	Good	Available
502	Sony Tripod	Sony	9.99	Good	Available
503	LED Light Kit	Generic	19.99	Good	Unavailable

CustomerID: 1
EquipmentID: 501
Status: Active

RENTALID	CUSTOMERID	EQUIPMENTID	STATUS	START_DATE	RETURN_DATE	TOTAL_COST
9001	1	501	Active	2025-11-10 0...		149.97
9002	2	502	Returned	2025-10-20 0...	2025-10-22 0...	19.98
9003	3	503	Active	2025-11-13 0...		39.98

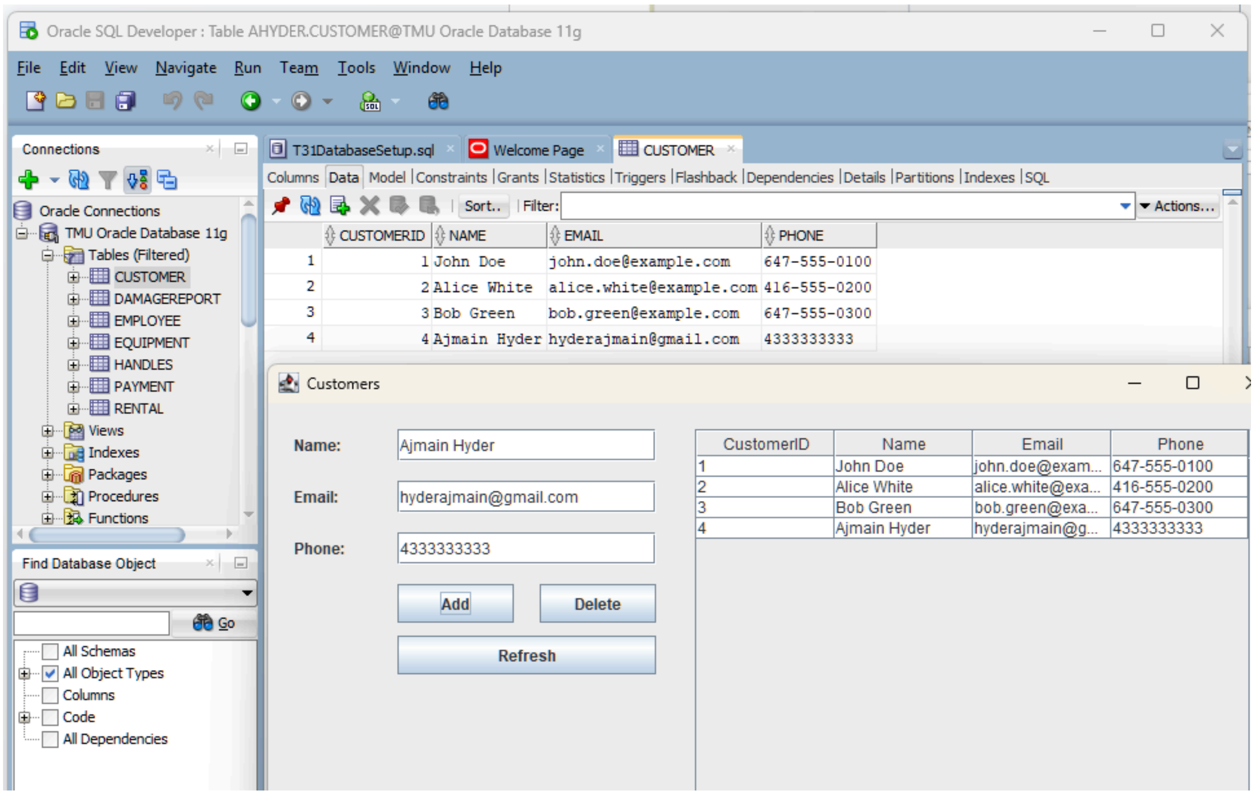
RentalID: 9001

PAYMENTID	RENTALID	AMOUNT	METHODUS...	PAYMENTD...
7001	9002	19.98	Credit Card	2025-10-20 ...
7002	9001	149.97	Credit Card	2025-11-11 ...

RentalID: 9001
EquipmentID: 501

REPORTID	RENTALID	EQUIPMENTID	DESCRIPTION	COST	REPORTDATE
8001	9002	502	Scratched leg	9.99	2025-10-22 0...

- UI of all possible management


Oracle SQL Developer : Table AHYDER.CUSTOMER@TMU Oracle Database 11g

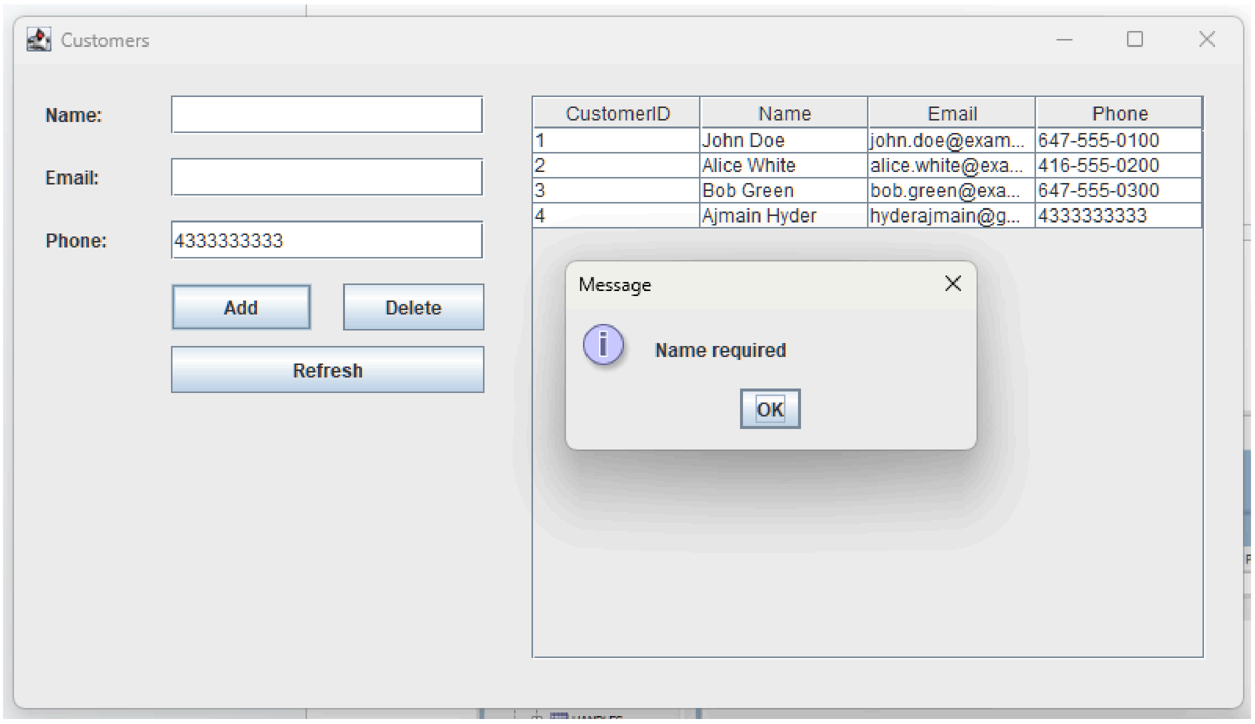
	CUSTOMERID	NAME	EMAIL	PHONE
1	1	John Doe	john.doe@example.com	647-555-0100
2	2	Alice White	alice.white@example.com	416-555-0200
3	3	Bob Green	bob.green@example.com	647-555-0300
4	4	Ajmain Hyder	hyderajmain@gmail.com	4333333333

Customers

Name:	Ajmain Hyder
Email:	hyderajmain@gmail.com
Phone:	4333333333

CustomerID	Name	Email	Phone
1	John Doe	john.doe@exam...	647-555-0100
2	Alice White	alice.white@exa...	416-555-0200
3	Bob Green	bob.green@exa...	647-555-0300
4	Ajmain Hyder	hyderajmain@g...	4333333333

- Customer added Successfully


Customers

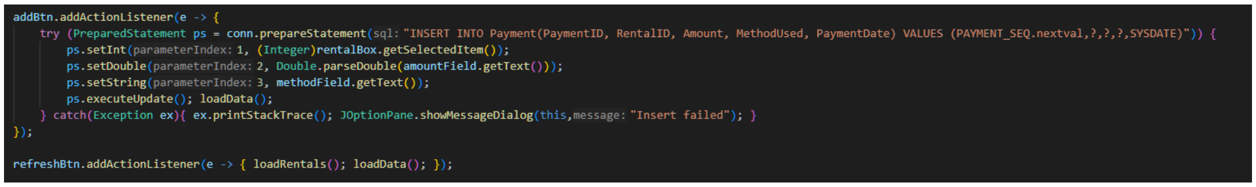
Name:	
Email:	
Phone:	4333333333

CustomerID	Name	Email	Phone
1	John Doe	john.doe@exam...	647-555-0100
2	Alice White	alice.white@exa...	416-555-0200
3	Bob Green	bob.green@exa...	647-555-0300
4	Ajmain Hyder	hyderajmain@g...	4333333333

Message

Name required

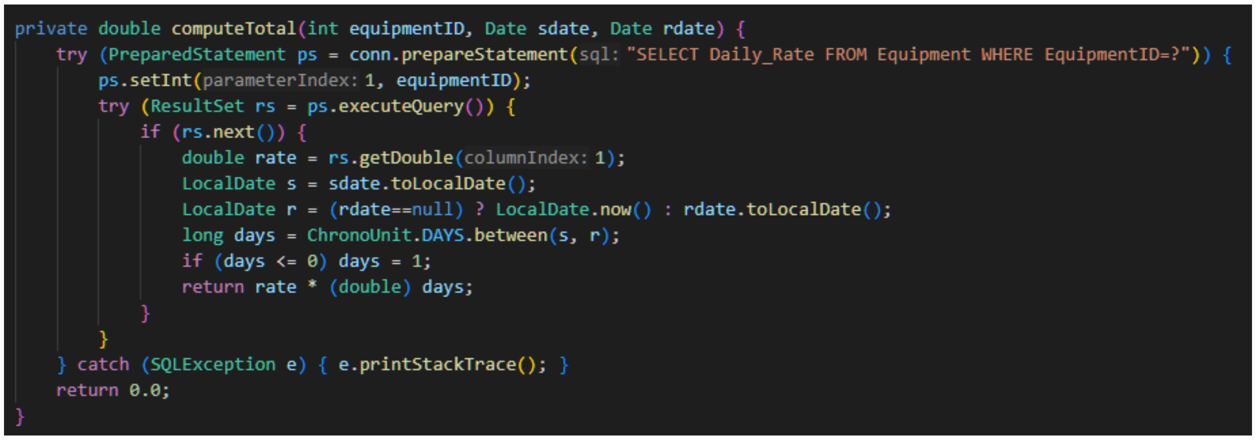
- Invalid Entries Example



```java
addBtn.addActionListener(e -> {
    try (PreparedStatement ps = conn.prepareStatement(sql: "INSERT INTO Payment(PaymentID, RentalID, Amount, MethodUsed, PaymentDate) VALUES (PAYMENT_SEQ.nextval,?,?,?,SYSDATE)")) {
        ps.setInt(parameterIndex: 1, (Integer)rentalBox.getSelectedItem());
        ps.setDouble(parameterIndex: 2, Double.parseDouble(amountField.getText()));
        ps.setString(parameterIndex: 3, methodField.getText());
        ps.executeUpdate(); loadData();
    } catch(Exception ex){ ex.printStackTrace(); JOptionPane.showMessageDialog(this,message: "Insert failed"); }
});

refreshBtn.addActionListener(e -> { loadRentals(); loadData(); });
```

- Example of insertion into a table



```java
private double computeTotal(int equipmentID, Date sdate, Date rdate) {
    try (PreparedStatement ps = conn.prepareStatement(sql: "SELECT Daily_Rate FROM Equipment WHERE EquipmentID=?")) {
        ps.setInt(parameterIndex: 1, equipmentID);
        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                double rate = rs.getDouble(columnIndex: 1);
                LocalDate s = sdate.toLocalDate();
                LocalDate r = (rdate==null) ? LocalDate.now() : rdate.toLocalDate();
                long days = ChronoUnit.DAYS.between(s, r);
                if (days <= 0) days = 1;
                return rate * (double) days;
            }
        }
    } catch (SQLException e) { e.printStackTrace(); }
    return 0.0;
}
```

- Queries Examples

Conclusion

This proposal outlines a Photo/Video Equipment Rental DBMS that manages customers, rentals, equipment, payments, staff, and suppliers. By designing entities and relationships carefully, the database will ensure efficient rental operations, accurate financial tracking, and improved customer service. This foundation can be extended to future features such as online booking or loyalty programs.