

OHIO STATE UNIVERSITY  
CSE 3242:Introduction to Database Systems

**Class Project Final Report**

Blake Theis, Christopher McDevitt, Matthew DiSanto, Sam Schmitt

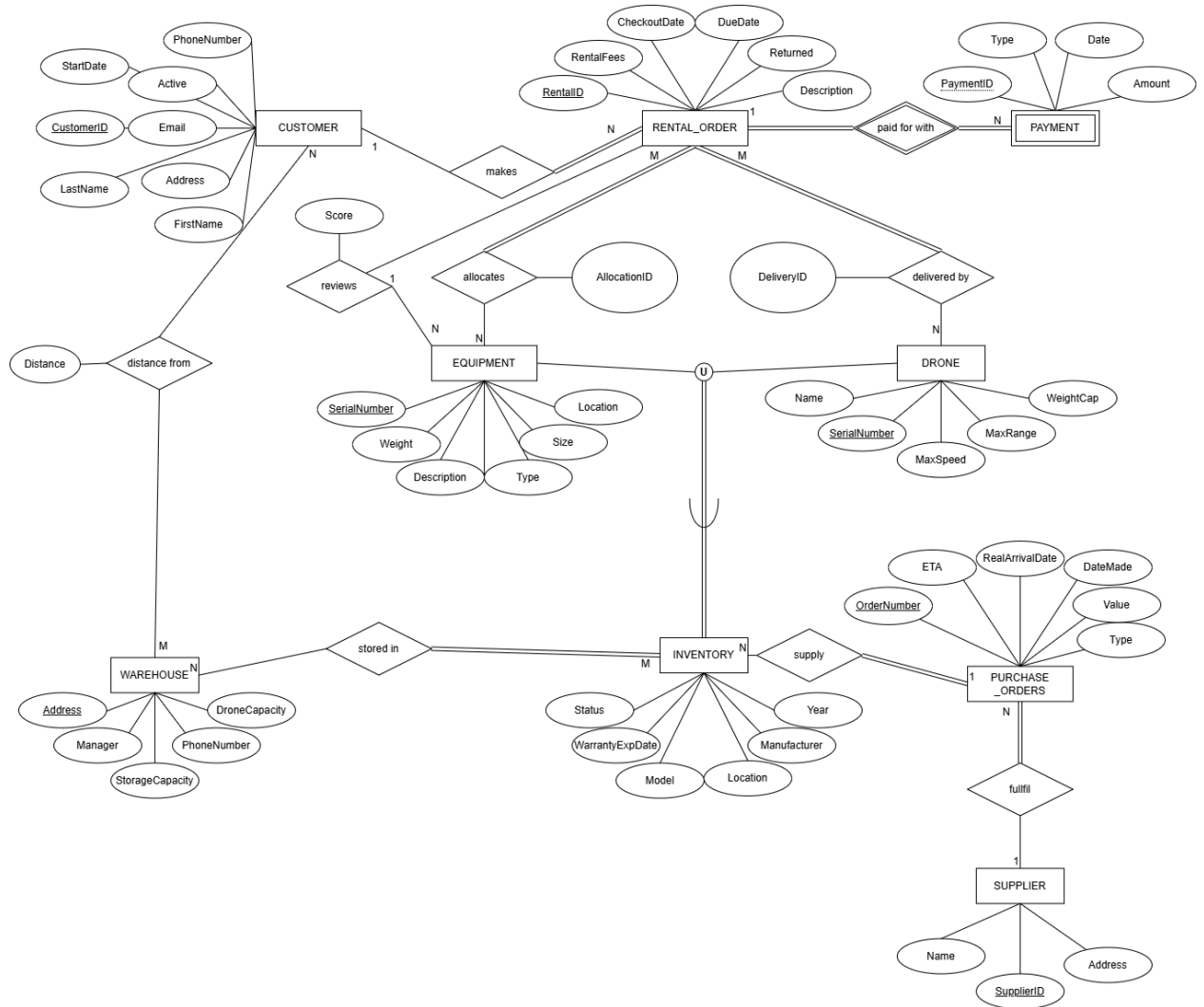
December 3, 2024

# Table Of Contents

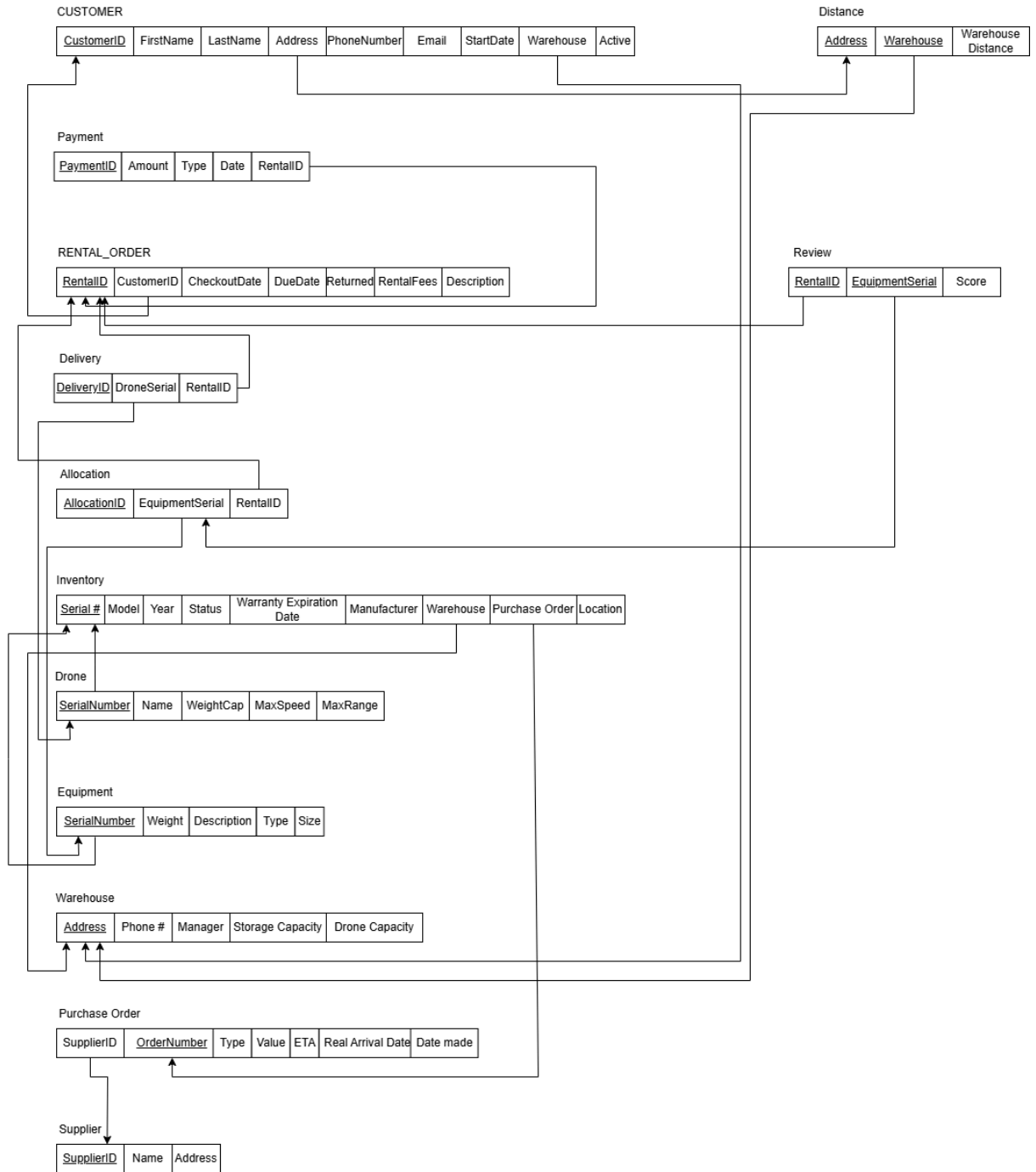
<b>Class Project Final Report.....</b>	<b>1</b>
<b>Table Of Contents.....</b>	<b>2</b>
Database Description.....	3
ER Model.....	3
Relational Schema.....	4
Normalization.....	5
Indexes.....	8
Views.....	9
Transactions.....	11
User Manual.....	13
Table Description.....	13
SQL Queries.....	28
INSERT Samples.....	42
Delete Samples.....	45
The SQL Database.....	47
Section 1: Database Files.....	47
Section 2: SQL scripts.....	48
Section 3: Program.....	49

# Database Description

## ER Model



# Relational Schema



## Normalization

### **Customer (BCNF):**

$\text{CustomerID} \rightarrow \{\text{FirstName, LastName, Address, PhoneNumber, Email, StartDate, Warehouse, Active}\}$

$\text{Email} \rightarrow \{\text{CustomerID, FirstName, LastName, Address, PhoneNumber, StartDate, Warehouse, Active}\}$

Customer has two functional dependencies, that CustomerID identifies the customer's name, address, phone, email, start date, which warehouse they receive items from, and if they are still an active customer. Email also uniquely identifies all other attributes including the Customer ID, as it is the data field used to create new accounts. There are no other functional dependencies, because there is no physical, legal, or organizational requirement that two accounts cannot have identical information in any or all fields. Either two people having the same information or by or a customer holding two accounts this is possible. The Customer table is in Boyce-Codd normal form because there are no nontrivial dependencies  $X \rightarrow A$  where  $X$  is non-prime.

### **Payment (BCNF):**

$\text{PaymentID} \rightarrow \{\text{Type, Date, Amount, RentalID}\}$

Payment has one functional dependency: its primary key, PaymentID, uniquely identifies all other attributes. There are no other functional dependencies because it is possible, if unlikely, that two payments could be made for the same rental, on the same day, using the same method for the same amount. Payment is in Boyce-Codd normal form because there are no nontrivial dependencies  $X \rightarrow A$  where  $X$  is non-prime.

### **Rental\_Order (BCNF):**

$\text{RentalID} \rightarrow \{\text{CustomerID, CheckoutDate, DueDate, Returned, RentalFees, Description}\}$

Rental Order has one functional dependency: its primary key, RentalID uniquely identifies all other attributes. There are no other functional dependencies because one customer could make two identical orders. Rental Order is in Boyce-Codd normal form because there are no nontrivial dependencies  $X \rightarrow A$  where  $X$  is non-prime.

### **Review (BCNF):**

$\{\text{RentalID, EquipmentSerial}\} \rightarrow \text{Score}$

Review has one functional dependency: RentalID and EquipmentSerial uniquely identify the Score. This makes sense because for each time a piece of equipment is rented it can be given one rating, and the ratings in 0-10 are not unique.. The Review table table is in Boyce-Codd normal form because there are no nontrivial dependencies  $X \rightarrow A$  where  $X$  is non-prime.

**Delivery (BCNF):**
$$\text{DeliveryID} \rightarrow \{\text{DroneSerial}, \text{RentalID}\}$$
$$\{\text{DroneSerial}, \text{RentalID}\} \rightarrow \text{DeliveryID}$$

There are two functional dependencies in the Delivery table. There are no other sensible functional dependencies as drones can deliver for multiple deliveries and a single delivery can have multiple drones. Delivery is in Boyce-Codd normal form because there are no nontrivial dependencies  $X \rightarrow A$  where  $X$  is non-prime.

**Allocation (BCNF):**
$$\text{AllocationID} \rightarrow \{\text{EquipmentSerial}, \text{RentalID}\}$$
$$\{\text{EquipmentSerial}, \text{RentalID}\} \rightarrow \text{AllocationID}$$

There are two functional dependencies in the Allocation table. The left sides of both functional dependencies are superkeys. There are no other functional dependencies as equipment can be rented many times and a rental can have many equipment being rented out. Allocation is in Boyce-Codd normal form because there are no nontrivial dependencies  $X \rightarrow A$  where  $X$  is non-prime.

**Inventory (BCNF):**
$$\text{SerialNumber} \rightarrow \{\text{Model}, \text{Year}, \text{Status}, \text{WarrantyExpiration}, \text{Manufacturer}, \text{Warehouse}, \text{PurchaseOrder}, \text{Location}\}$$

Inventory has one function dependency, that the Serial Number of the inventory item uniquely all other attributes. Note that the Inventory table is the union of the Drone and Equipment tables. Inventory is in Boyce-Codd normal form because there are no nontrivial dependencies  $X \rightarrow A$  where  $X$  is non-prime.

**Drone (BCNF):**
$$\text{SerialNumber} \rightarrow \{\text{Name}, \text{WeightCap}, \text{MaxSpeed}, \text{MaxRange}\}$$

The Drone table has a single functional dependency. Its unique serial number uniquely determines its attributes. Names are not required to be unique and different drones can have the same characteristics. Drone is in Boyce-Codd normal form because there are no nontrivial dependencies  $X \rightarrow A$  where  $X$  is non-prime.

**Equipment (BCNF):**
$$\text{SerialNumber} \rightarrow \{\text{Weight}, \text{Description}, \text{Type}, \text{Size}\}$$

The Equipment table has a single functional dependency. Its unique serial number functionally determines the weight, description, type and size of the equipment item. The Equipment table is in Boyce-Codd normal form because there are no nontrivial dependencies  $X \rightarrow A$  where  $X$  is non-prime.

**Warehouse (BCNF):**

Address  $\rightarrow$  {PhoneNumber, Manager, StorageCapacity, DroneCapacity}

The Warehouse table has a single functional dependency: address, the primary key, identifies all other attributes. Multiple warehouses may share a phone number and/or manager. Storage capacity and drone capacity are clearly not required to be unique, so there are no other functional dependencies. The Warehouse table is in Boyce-Codd normal form because there are no nontrivial dependencies  $X \rightarrow A$  where  $X$  is non-prime.

**Purchase Order (BCNF):**

OrderNumber  $\rightarrow$  {SupplierID, Type, Value, ETA, RealArrivalDate, DateMade}

The Purchase Order table has a single functional dependency. The primary key, OrderNum, determines all other attributes. There are no other functional dependencies, it is not impossible to make two identical orders. The Purchase Order table is in Boyce-Codd normal form because there are no nontrivial dependencies  $X \rightarrow A$  where  $X$  is non-prime.

**Supplier (BCNF):**

SupplierID  $\rightarrow$  {Name, Address}

{Name, Address}  $\rightarrow$  SupplierID

The Supplier table has two functional dependencies, the supplier id uniquely identifies the name and address of the supplier and the name and address of the supplier uniquely identifies the supplierID. If a supplier that the company order's from has the same name and address, it is in fact the same supplier from the company's perspective. Supplier is in Boyce-Codd normal form because there are no nontrivial dependencies  $X \rightarrow A$  where  $X$  is non-prime.

## Indexes

### **Index 1: RentalOrder.CustomerID**

We use an index on RentalOrder.CustomerID. While this index will be updated whenever an order is created, it will be used incredibly frequently because nearly all queries involving a customer require joining with RentalOrder on CustomerID. For example, a customer viewing their order history.

### **Index 2: RentalOrder.Returned**

We use an index on RentalOrder.Returned. This index is updated relatively infrequently and it is used in the overdueRentals view and in similar administrative queries.

### **Index 3: Allocation.EquipmentSerial**

We use an index on Allocation.EquipmentSerial. This index is typically updated whenever an order is created (potentially multiple times per order), but is used very often also to link a rental order with the equipment that was rented.

### **Index 4: Allocation.RentalID**

We use an index on Allocation.EquipmentSerial. This index is updated when an allocation is created (potentially multiple times per order), but it is used very frequently to link a rental order with the equipment that was rented.



## Views

### View 1 - Daily Rentals

- I. A view shows the total number of rental orders made each day along with the total value of the equipment rented that day. This would allow administrators to quickly see the changes in how the organization is doing.

- II. 
$$\text{data} = (((\text{RentalOrder} \bowtie_{\text{RentalOrder.RentalID}=\text{Allocation.RentalID}} \text{Allocation}) \bowtie_{\text{Allocation.EquipmentSerial}=\text{Inventory.SerialNumber}} \text{Inventory}) \bowtie_{\text{PurchaseOrder.OrderNumber}=\text{Inventory.PurchaseOrder}} \text{PurchaseOrder})$$

- III. 
$$\text{count(RentalOrder.RentalID), sum(value)} \int_{\text{RentalOrder.CheckoutDate}/84600} (\text{data})$$
  
 CREATE VIEW dailyRentals AS  
 SELECT RentalOrder.CheckoutDate / 84600,  
 count(RentalOrder.RentalID), sum(value)  
 FROM RentalOrder, Allocation, Inventory, PurchaseOrder  
 WHERE RentalOrder.RentalID = Allocation.RentalID AND  
 Allocation.EquipmentSerial = Inventory.SerialNumber AND  
 PurchaseOrder.OrderNumber = Inventory.PurchaseOrder  
 GROUP BY RentalOrder.CheckoutDate / 84600

	RentalOrder.CheckoutDate / 84600	count(RentalOrder.RentalID)	sum(value)
1	20452	1	1707
2	20080424	1	1496
3	20092289	1	1496
4	20094387	1	858
5	20150103	2	1716
6	20170368	1	1966
7	20214662	1	1707
8	20219899	1	1966
9	20236486	3	2808
10	20242308	1	746
11	20282959	2	1854
12	20329711	1	1707
13	20334091	1	1831
14	20435739	3	2574

- IV.

## View 2 Overdue Rentals

- I. A view that shows the customers with rentals that are currently past due and the value of all equipment that is out past the due date. This will be useful in prioritizing which people we need to get stuff back from or canceling memberships of those who don't return things.

II.  $CustomerID \bowtie_{sum(RentalFees)} (\sigma_{Returned=0 \text{ AND } DueDate < unixepoch("now")} (Customer * RentalOrder))$

III. CREATE VIEW overdueRentals AS  
SELECT Customer.CustomerID as CustomerID, sum(RentalFees) FROM  
(Customer JOIN RentalOrder ON  
Customer.CustomerID=RentalOrder.CustomerID)  
WHERE Returned=0 AND DueDate < unixepoch("now")  
GROUP BY Customer.CustomerID;

	CustomerID	sum(RentalFees)	DueDate
1	1	60	1715309081
2	2	40	1719353765
3	3	77	1726788180
4	6	44	1729424780
5	7	46	1729431102
6	11	108	1719636015
7	12	71	1729888943
8	13	33	1727254911
9	16	48	1718353818
10	19	67	1729593248
11	20	43	1725396010

IV.

## Transactions

A transaction is needed to ensure the database is not left in an inconsistent state. If an operation is split into multiple SQL queries, a failure or crash after one of them may leave the database in an inconsistent state. Below are three sample transactions for our database.

### Transaction 1: Create a rental order (all allocations and deliveries).

```
BEGIN TRANSACTION makeorder;
INSERT INTO RentalOrder VALUES ((SELECT max(RentalID) + 1 FROM
RentalOrder), 1, unixepoch('now'), unixepoch('now') + 60*60*24*5,
0, 50.51, 'small inflatable movie screen', (SELECT Address FROM
Customer WHERE CustomerID=1));
INSERT INTO Allocation VALUES ((SELECT max(AllocationID) + 1 FROM
Allocation), 90036, (SELECT max(RentalID) FROM RentalOrder));
END TRANSACTION;
```

This transaction represents a customer renting one small inflatable movie screen. To do this a rental order is added to the database and a piece of equipment is allocated to said order. A transaction is needed for this operation to ensure that the database does not record an incomplete order.

### Transaction 2: Adding a new drone to the system.

```
BEGIN TRANSACTION addDrone;
INSERT INTO Inventory VALUES (84324, 'Viper', 2027, 'active',
unixepoch('now')+60*60*24*365*6, 'Raytheon', '75 Carberry Road',
20, 'unknown');
INSERT INTO Drone VALUES (84324, 'Missal man', 550, 17500, 6300);
END TRANSACTION;
```

A transaction is needed for this operation to ensure that the inventory and drone tables are consistent with each other. Without a transaction it is possible that a drone could be entered into the inventory table, but not the drone table, or the drone table, but not the inventory table. In either case, the data is invalid and some queries may give incorrect results.

**Transaction 3: Updating a customer's address**

```
BEGIN TRANSACTION move_customer;  
INSERT OR IGNORE INTO Distance VALUES ("1234 Place Road", "75  
Carberry Road", 144);  
UPDATE OR ROLLBACK Customer SET Address="1234 Place Road",  
Warehouse="75 Carberry Road" WHERE CustomerID=1;  
COMMIT;
```

A transaction is needed for this operation to ensure that the distance and customer tables are consistent with each other. The distance between the new address and the warehouse is added first, and then the customer's address is set to the new address.

# User Manual

## Table Description

### **Allocation**

The entries in the Allocation table represent the assignment of rental equipment to a rental order.

Attributes:

1. AllocationID
  - a. Description: A unique number used to identify an allocation.
  - b. Type: INTEGER
  - c. Constraints: PRIMARY KEY, NOT NULL
2. EquipmentSerial
  - a. Description: The serial number of the allocated equipment.
  - b. Type: TEXT
  - c. Constraints: FOREIGN KEY referencing Equipment.SerialNumber, NOT NULL
3. RentalID
  - a. Description: The ID of the rental this allocation was made for.
  - b. Type: INTEGER
  - c. Constraints: FOREIGN KEY referencing RentalOrder.RentalID, NOT NULL

### **Customer**

Each row in the Customer table represents a customer: an individual who rents equipment from the service. Customers must provide their name, address, phone number, and email to sign up for service. Also stored is the customer's associated warehouse, when their account was created, and if they are still an active customer or not.

Attributes:

1. CustomerID
  - a. Description: A unique number used to identify a customer.
  - b. Type: INTEGER
  - c. Constraints: PRIMARY KEY, NOT NULL
2. FirstName
  - a. Description: The customer's first name.
  - b. Type: TEXT
  - c. Constraints: NOT NULL
3. LastName
  - a. Description: The customer's last name.
  - b. Type: TEXT
  - c. Constraints: NOT NULL
4. Address
  - a. Description: The customer's address
  - b. Type: TEXT
  - c. Constraints: part of a FOREIGN KEY referencing (Distance.Address, Distance.Warehouse), NOT NULL
5. Phone Number
  - a. Description: The customer's phone number.
  - b. Type: TEXT
  - c. Constraints: NOT NULL
6. Email
  - a. Description: The customer's email address
  - b. type: TEXT

- c. Constraints, UNIQUE, NOT NULL
- 7. StartDate
  - a. Description: The date and time the customer signed up to the service.
  - b. Type: INTEGER (Unix Timestamp)
  - c. Constraints: NOT NULL
- 8. Warehouse
  - a. Description: The nearest warehouse to the customer.
  - b. Type: TEXT
  - c. Constraints: part of a FOREIGN KEY referencing (Distance.Address, Distance.Warehouse), NOT NULL
- 9. Active
  - a. Description: Whether or not the customer is still an active user of the service. The value is 1 if they are active, 0 if not.
  - b. Type: INTEGER
  - c. Constraints: NOT NULL

## **Delivery**

Each entry in a Delivery table records the usage of a drone for delivering equipment for a particular rental.

Attributes:

- 1. DeliveryID
  - a. Description: A unique number identifying a delivery.
  - b. Type: INTEGER
  - c. Constraints: PRIMARY KEY, NOT NULL
- 2. DroneSerial
  - a. Description: The serial number of the drone used for this delivery.

- b. Type: TEXT
  - c. Constraints: FOREIGN KEY referencing Drone.SerialNumber, NOT NULL
- 3. RentalID
  - a. Description: The rental order this delivery is for.
  - b. Type: INTEGER
  - c. Constraints: FOREIGN KEY referencing RentalOrder.RentalID, NOT NULL

## **Distance**

Each row in the Distance table represents the distance between a warehouse and a customer's address.

### Attributes

- 1. Address
  - a. Description: The address used in the distance calculation.
  - b. Type: TEXT
  - c. Constraints: PRIMARY KEY, NOT NULL
- 2. Warehouse
  - a. Description: The warehouse used in the distance calculation.
  - b. Type: TEXT
  - c. Constraints: PRIMARY KEY, NOT NULL
- 3. WarehouseDistance
  - a. Description: The distance between the address and the warehouse, in miles.
  - b. Type: INTEGER
  - c. Constraints: NOT NULL



## Drone

Each row in the Drone table represents the drone-specific characteristics of a particular drone. All drones are also in the Inventory table, which records common characteristics between drones and equipment. Recorded attributes include the drone's name, weight capacity, maximum speed, and range. Drones are identified by their manufacturer issued serial number.

### Attributes:

1. SerialNumber
  - a. Description: The serial number of the drone.
  - b. Type: TEXT
  - c. Constraints: PRIMARY KEY, FOREIGN KEY referencing Equipment.SerialNumber
2. Name
  - a. Description: The name of the drone.
  - b. Type: TEXT
  - c. Constraint: NOT NULL
3. WeightCap
  - a. Description: The maximum weight the drone can carry, in pounds.
  - b. Type: INTEGER
  - c. Constraint: NOT NULL
4. MaxSpeed
  - a. Description: The maximum speed of the drone.
  - b. Type: INTEGER
  - c. Constraint: NOT NULL
5. MaxRange
  - a. Description: The maximum range of the drone, in miles.
  - b. Type: INTEGER

- c. Constraint: NOT NULL

## Equipment

Each row in the Equipment table represents the equipment-specific characteristics of a particular piece of rentable equipment. All equipment items are also in the Inventory table, which records common characteristics between drones and equipment. Included attributes for each item include weight, description, type, size, and a short description. Equipment is identified by its manufacturer issued serial number.

### Attributes:

1. SerialNumber
  - a. Description: The serial number of the equipment item.
  - b. Type: TEXT
  - c. Constraints: PRIMARY KEY, FOREIGN KEY referencing Inventory.SerialNumber, NOT NULL
2. Weight
  - a. Description: The name of the equipment item.
  - b. Type: TEXT
  - c. Constraints: NOT NULL
3. Description
  - a. Description: The description of the equipment item.
  - b. Type: TEXT
  - c. Constraints: NOT NULL
4. Type
  - a. Description: The category of the equipment.
  - b. Type: INTEGER
  - c. Constraints: NOT NULL

## 5. Size

- a. Description: The size of the equipment in warehouse units.
- b. Type: INTEGER
- c. Constraints: NOT NULL

## Inventory

Each row in the Inventory table represents a drone or a piece of rentable equipment. The inventory table records information common to drones and equipment items. Drone and equipment specific information is found in the Drone and Equipment tables respectively.

Attributes:

### 1. SerialNumber

- a. Description: The serial number of the inventory item.
- b. Type: TEXT
- c. Constraints: PRIMARY KEY, NOT NULL

### 2. Model

- a. Description: The model of the inventory item.
- b. Type: TEXT
- c. Constraints: NOT NULL

### 3. Year

- a. Description: The description of the inventory item.
- b. Type: INTEGER
- c. Constraints: NOT NULL

### 4. Status

- a. Description: The status of the inventory item. For example: “active”, “offline”, or “ordered”

- b. Type: TEXT
  - c. Constraints: NOT NULL
- 5. WarrantyExpirationDate
  - a. Description: When this inventory item's warranty will expire.
  - b. Type: INTEGER (Unix Timestamp)
  - c. Constraints: NOT NULL
- 6. Manufacturer
  - a. Description: The manufacturer of this inventory item.
  - b. Type: TEXT
  - c. Constraints: NOT NULL
- 7. Warehouse
  - a. Description: The home warehouse of this inventory item.
  - b. Type: TEXT
  - c. Constraints: FOREIGN KEY referencing Warehouse.Address, NOT NULL
- 8. PurchaseOrder
  - a. Description: The order that purchased this item.
  - b. Type: INTEGER
  - c. Constraints: FOREIGN KEY referencing PurchaseOrder.OrderNumber, NOT NULL
- 9. Location
  - a. Description: The current location of the drone or equipment, or "unknown"
  - b. Type: TEXT
  - c. Constraints: NOT NULL

## **Payment**

Each row in the Payment table represents a payment made by a customer for a rental order. Included attributes include the amount paid, what type of payment method was used, the date the payment was made, and what rental order it is associated with.

#### Attributes

##### 1. PaymentID

- a. Description: A number that uniquely identifies the payment made by the customer.
- b. Type: INTEGER
- c. Constraints: PRIMARY KEY, NOT NULL

##### 2. Amount

- a. Description: The amount of money being paid in this payment by the customer for this rental.
- b. Type: INTEGER
- c. Constraints: NOT NULL

##### 3. Type

- a. Description: The type of payment method being used by the customer. (Credit Card, Check, Cash, etc)
- b. Type: INTEGER
- c. Constraints: NOT NULL

##### 4. Date

- a. Description: The date that the customer made the payment.
- b. Type: INTEGER (Unix Timestamp)
- c. Constraints: NOT NULL

##### 5. RentalID

- a. Description: The rental that is being paid for, in full or in part, by this payment.
- b. Type: INTEGER

- c. Constraints: FOREIGN KEY referencing RentalOrder.RentalID, NOT NULL

## **PurchaseOrder**

Each row in the PurchaseOrder table represents the purchase of equipment by the company. Included attributes include the which supplier equipment is being purchased from, the category of goods being purchased, the per item value of some items, the date the order was made, the eta for the delivery, and the real delivery date if applicable.

Attributes:

1. SupplierID
  - a. Description: A foreign key that identifies which supplier the company is purchasing from.
  - b. Type: INTEGER
  - c. Constraints: FOREIGN KEY referencing Supplier.SupplierID, NOT NULL
2. OrderNumber
  - a. Description: The purchase number of the order that uniquely identifies the purchase order.
  - b. Type: INTEGER
  - c. Constraints: PRIMARY KEY, NOT NULL
3. Type
  - a. Description: What category of goods are being purchased in the purchase order, office supplies, athletics supplies, cookware, etc.
  - b. Type: INTEGER
  - c. Constraints: NOT NULL
4. Value
  - a. Description: The value of the items that are being purchased with this order.
  - b. Type: INTEGER
  - c. Constraints: NOT NULL
5. ETA

- a. Description: The expected date that the items will be delivered.
  - b. Type: INTEGER (Unix Timestamp)
  - c. Constraints: NOT NULL
6. RealArrivalDate
- a. Description: The date that the items were actually delivered.
  - b. Type: INTEGER (Unix Timestamp)
  - c. Constraints: None
7. DateMade
- a. Description: The date that the purchase order was made.
  - b. Type: INTEGER (Unix Timestamp)
  - c. Constraints: NOT NULL

## **RentalOrder**

Each row in the RentalOrder table represents a user submitting a rental order of equipment by a customer. Attributes for each entry include the ID of the customer that made the order, when the order was made and is due, if the rented equipment has been returned, the amount owed for the rental, and special delivery instructions.

Attributes:

- 1. RentalID
  - a. Description: A number that uniquely identifies a rental order.
  - b. Type: INTEGER
  - c. Constraints: PRIMARY KEY, NOT NULL
- 2. CustomerID
  - a. Description: A foreign key that identifies which customer is placing the order.
  - b. Type: INTEGER
  - c. Constraints: FOREIGN KEY referencing Customer.CustomerID, NOT NULL

3. CheckoutDate

- a. Description: The day that the customer wants the rental to be delivered on.
- b. Type: INTEGER (Unix Timestamp)
- c. Constraints: NOT NULL

4. DueDate

- a. Description: The day the customer has to return the rental, before facing overdue charges.
- b. Type: INTEGER (Unix Timestamp)
- c. Constraints: NOT NULL

5. Returned

- a. Description: A boolean field that is true if the items from the rental have been returned.
- b. Type: INTEGER
- c. Constraints: NOT NULL

6. RentalFees

- a. Description: The price the customer is charged for their rental.
- b. Type: INTEGER
- c. Constraints: NOT NULL

7. Description

- a. Description: A short description of special delivery instructions written by the customer
- b. Type: TEXT
- c. Constraints: NOT NULL

**Review**

Each row in the Review table represents a customer's review of a rented equipment item.

Attributes:



1. RentalID

- a. Description: This identifies which rental order the review belongs to.
- b. Type: INTEGER
- c. Constraints: PRIMARY KEY, NOT NULL

2. EquipmentSerial

- a. Description: This identifies which piece of equipment from that rental order is being reviewed.
- b. Type: TEXT
- c. Constraints: PRIMARY KEY, NOT NULL

3. Score

- a. Description: The score given by the reviewing customer for the equipment that they rented.
- b. Type: TEXT
- c. Constraints: NOT NULL

## **Supplier**

Each row in the Supplier table represents a seller that the company purchased equipment from.

Attributes:

1. SupplierID

- a. Description: A unique number used to identify a supplier.
- b. Type: INTEGER
- c. Constraints: PRIMARY KEY, NOT NULL

2. Name

- a. Description: The name of the supplier. Either a legal name or a corporate name.

- b. Type: TEXT
  - c. Constraints: NOT NULL
- 3. Address
  - a. Description:
  - b. Type: TEXT
  - c. Constraints: NOT NULL

## **Warehouse**

Each row in the Warehouse table represents a location used by the company to store equipment and drones.

Attributes:

- 1. Address
  - a. Description: The street address that the warehouse is located at.
  - b. Type: TEXT
  - c. Constraints: PRIMARY KEY, NOT NULL
- 2. PhoneNumber
  - a. Description: The phone number that customers and employees can call to contact the warehouse.
  - b. Type: TEXT
  - c. Constraints: NOT NULL
- 3. Manager
  - a. Description: The name of the warehouse manager.
  - b. Type: TEXT
  - c. Constraints: NOT NULL

4. StorageCapacity

- a. Description: The amount of space available for item storage in the warehouse.
- b. Type: INTEGER
- c. Constraints: NOT NULL

5. DroneCapacity

- a. Description: The number of drones that the warehouse can contain/utilized.
- b. Type: INTEGER
- c. Constraints: NOT NULL

## SQL Queries

1.

- I. Find the type of all Equipment by MANUFACTURER released after YEAR (you choose how to designate the manufacturer, year)
- II.  $\pi_{\text{Type}}(\sigma_{\text{year} > \text{YEAR and Manufacturer} == \text{MANUFACTURER}}(\text{EQUIPMENT} * \text{INVENTORY}))$
- III. `SELECT Type FROM (Inventory NATURAL JOIN Equipment) WHERE Manufacturer="AMD" AND Year > 2017;`
- IV. STATUS: Tested and correct

1 `SELECT Type FROM (INVENTORY NATURAL JOIN EQUIPMENT) WHERE Manufacturer="AMD" AND Year > 2017;`

Grid view Form view

Total rows loaded: 2

Type
7
5

V.

2.

- I. Give all the equipment and the date(s) of their checkout from a single MEMBER (you choose how to designate the member).
- II. `orders = RENTAL_ORDER *  $\sigma_{\text{CustomerID} == \text{MEMBER}}$ (CUSTOMER)`
- III.  $\pi_{\text{Serial\#, Checkout\_Date}}(\text{orders} * \text{ALLOCATION})$   
`SELECT EquipmentSerial, CheckoutDate FROM ((Customer JOIN RentalOrder ON Customer.CustomerID=RentalOrder.CustomerID) NATURAL JOIN Allocation) WHERE FirstName="Addy" AND LastName="Clissold";`
- IV. STATUS: Tested and correct

1 `SELECT EquipmentSerial, CheckoutDate FROM ((Customer JOIN RentalOrder ON Customer.CustomerID=RentalOrder.CustomerID) NATURAL JOIN Allocation)`  
 2 `WHERE FirstName="Addy" AND LastName="Clissold";`

Grid view Form view

Total rows loaded: 1

EquipmentSerial	CheckoutDate
78619	1712499258

V.

3.

I. List all drones by weight capacity and its unique identifiers with 2 or less units held by a WAREHOUSE (you choose how to designate the warehouse).

II.  $w = \rho_{(warehouse,...)}(\sigma_{City=CITY}(Warehouse))$

$\pi_{weight\_capacity,serial\#}(\sigma_{model\_count \leq 2}(f_{Count Model}(w * (Drone * Inventory))))$

III. `SELECT Model, WeightCap, count(Model) FROM (Drone NATURAL JOIN Inventory) WHERE Warehouse="324 Magdeline Avenue" GROUP BY Model HAVING count(Model) <= 2;`

IV. STATUS: Tested and correct

```
1 SELECT Model, WeightCap, count(Model) FROM (Drone NATURAL JOIN Inventory)
2 WHERE Warehouse="324 Magdeline Avenue" GROUP BY Model HAVING count(Model) <= 2;
```

Grid view		Form view	
Total rows loaded: 2			
Model	WeightCap	count(Model)	
1 Fancy	38	1	
2 Speedy	48	1	

V.

4.

I. Give all the members that live more than 15 miles away from the warehouse who checked out equipment delivered by a particular DRONE (you choose how to designate the drone), as well as the equipment they checked out.

II.  $orders = \sigma_{Warehouse\_Distance > 15}(CUSTOMER) * RENTAL\_ORDER$

$deliveries = orders * allocation * \sigma_{DroneSerial=SDRONE}(DELIVERY)$

$\pi_{customerID,First\_Name,Last\_Name,EquipmentSerial}(deliveries)$

III. `SELECT FirstName, LastName, EquipmentSerial FROM (Customer JOIN RentalOrder ON Customer.CustomerID=RentalOrder.CustomerID) NATURAL JOIN Distance NATURAL JOIN Delivery NATURAL JOIN Allocation WHERE WarehouseDistance > 15 AND DroneSerial="71576";`

IV. STATUS: Tested and correct

```

1 SELECT FirstName, LastName, EquipmentSerial FROM (Customer JOIN RentalOrder ON Customer.CustomerID=RentalOrder.CustomerID)
2 NATURAL JOIN Distance NATURAL JOIN Delivery NATURAL JOIN Allocation
3 WHERE WarehouseDistance > 15 AND DroneSerial="71576";

```

Grid view Form view

Total rows loaded: 1

	FirstName	LastName	EquipmentSerial
1	Lindsay	Wagenen	49889

V.

5.

I. Find the total number of unique drones that have picked up items checked out on certain Address (you choose how to designate the member address).

II.  $\text{member} = \sigma_{\text{address}=\$ADDRESS}(\text{CUSTOMER})$

$\text{rentals} = \text{RENTAL\_ORDER} * \text{member}$

$\text{drones} = \text{rentals} * \text{DELIVERY}$

$f_{\text{COUNT}(\text{DroneSerial})}(\pi_{\text{DroneSerial}}(\text{drones}))$

III. 

```
SELECT count(distinct(DroneSerial)) FROM (Customer JOIN
RentalOrder ON Customer.CustomerID=RentalOrder.CustomerID)
NATURAL JOIN Delivery WHERE Customer.Address="572 Bayside
Terrace" GROUP BY Customer.Address;
```

IV. STATUS: Tested and correct

```

1 SELECT count(distinct(DroneSerial)) FROM (Customer JOIN RentalOrder ON Customer.CustomerID=RentalOrder.CustomerID) NATURAL JOIN Delivery
2 WHERE Customer.Address="78804 Petterle Way" GROUP BY Customer.Address;

```

Grid view Form view

Total rows loaded: 1

	count(DISTINCT (DroneSerial))
1	1

V.

6.

I. Find the member who has checked out the most equipment and the total number of items they have checked out.

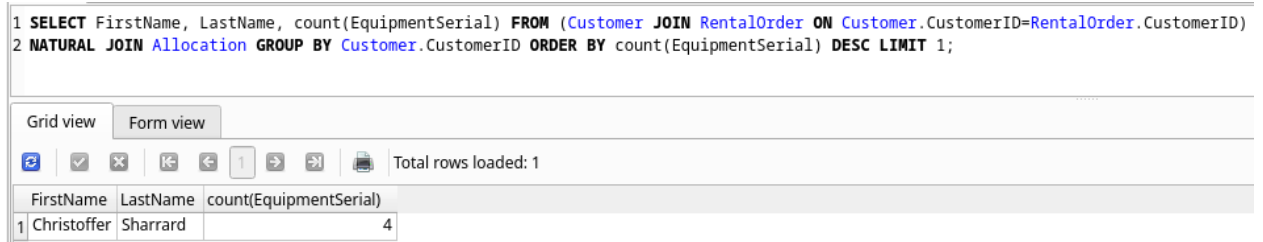
II.  $\text{equipment\_counts} = \pi_{\text{CustomerID}} f_{\text{COUNT EquipmentSerial}}(\text{RENTAL\_ORDER} * \text{ALLOCATION})$

$\text{max\_count} = f_{\text{MAX count\_EquipmentSerial}}(\text{equipment\_counts})$

$\text{member} = \pi_{\text{CustomerID}}(\sigma_{\text{count\_EquipmentSerial}=\text{max\_count}}(\text{equipment\_counts}))$

III. `SELECT FirstName, LastName, count(EquipmentSerial) FROM (Customer JOIN RentalOrder ON Customer.CustomerID=RentalOrder.CustomerID) NATURAL JOIN Allocation GROUP BY Customer.CustomerID ORDER BY count(EquipmentSerial) DESC LIMIT 1;`

IV. STATUS: Tested and correct



The screenshot shows a database query interface. At the top, the SQL query is displayed: `1 SELECT FirstName, LastName, count(EquipmentSerial) FROM (Customer JOIN RentalOrder ON Customer.CustomerID=RentalOrder.CustomerID) 2 NATURAL JOIN Allocation GROUP BY Customer.CustomerID ORDER BY count(EquipmentSerial) DESC LIMIT 1;`. Below the query, there are tabs for 'Grid view' and 'Form view'. The 'Grid view' is selected, and it shows a table with the following data:

	FirstName	LastName	count(EquipmentSerial)
1	Christoffer	Sharrard	4

Below the table, there is a status bar that says 'Total rows loaded: 1'.

V.

7.

I. Find the total quantity of Equipment purchases in the last 6 months by equipment type, provide the total dollar amount for each type and the average weight.

II.  $\text{RecentOrders} \leftarrow \sigma_{\text{DateMade} > (\text{CURRENT\_DATE} - 6 \text{ months})}(\text{PurchaseOrder})$

$\text{OrderItems} \leftarrow \text{RecentOrders} \bowtie_{\text{OrderNum} = \text{Purchase\_Order}}(\text{Inventory})$

$\text{EquipDetails} \leftarrow \text{OrderItems} \bowtie_{\text{Serial\#} = \text{Serial\#}}(\text{Equipment})$

$\text{Result} \leftarrow f_{\text{Type}, \text{COUNT}(\text{Serial\#}), \text{SUM}(\text{Value}), \text{AVG}(\text{Weight})}(\text{EquipDetails})$

III. `SELECT count(Inventory.SerialNumber), sum(Value), avg(Weight) FROM PurchaseOrder JOIN (Inventory JOIN Equipment on Inventory.SerialNumber=Equipment.SerialNumber) ON PurchaseOrder.OrderNumber=Inventory.PurchaseOrder WHERE ((unixepoch("now") - DateMade) < (6 * 28 * 24 * 60 * 60)) GROUP BY Equipment.Type;`

IV. STATUS: Tested and correct

```

1 SELECT count(Inventory.SerialNumber), sum(Value), avg(Weight) FROM PurchaseOrder JOIN
2 (Inventory JOIN Equipment on Inventory.SerialNumber=Equipment.SerialNumber)
3 ON PurchaseOrder.OrderNumber=Inventory.PurchaseOrder
4 WHERE ((unixepoch("now") - DateMade) < (6 * 28 * 24 * 60 * 60)) GROUP BY Equipment.Type;

```

Grid view		Form view	
		Total rows loaded: 6	
	count(Inventory.SerialNumber)	sum(Value)	avg(Weight)
1	1	1707	12
2	2	2565	38.5
3	4	5375	45.75
4	3	2526	57.66666666666666
5	3	3682	76.66666666666667
6	1	746	75

V.

## Extra Queries

1.

I. Determine which warehouse contains the most inventory.

II.  $\text{counts} = \text{Address} \bowtie \text{COUNT Equipment.SerialNumber}(\rho_{(\text{warehouse}, \dots)}(\text{Warehouse}) * (\text{Inventory} * \text{Equipment}))$

$\pi_{\text{Address}}(f_{\text{MAX count\_equipment}}(\text{warehouse}))$

III. SELECT Address FROM Warehouse JOIN  
(Equipment JOIN Inventory ON  
Equipment.SerialNumber=Inventory.SerialNumber) ON  
Warehouse=Address GROUP BY Address ORDER BY  
count(Equipment.SerialNumber) DESC LIMIT 1;

IV. STATUS: Tested and Correct.

```

1 SELECT Address FROM Warehouse JOIN
2 (Equipment JOIN Inventory ON Equipment.SerialNumber=Inventory.SerialNumber) ON Warehouse=Address GROUP BY Address ORDER BY count(Equipment.SerialNumber) DESC LIMIT 1;

```

Grid view

Form view

<

V.

2.

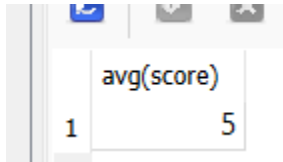
I. Get the average rating given by a customer

II.  $\text{orders} = \sigma_{\text{CustomerID}=12}(\text{CUSTOMER} * \text{RENTAL\_ORDER} * \text{REVIEW})$



- III.  $f_{AVG\ Score}(orders)$   
 SELECT avg(score) FROM (Customer JOIN RentalOrder ON  
 Customer.CustomerID=RentalOrder.CustomerID) NATURAL JOIN Review  
 WHERE Customer.CustomerID=12;

IV. STATUS: Tested and Correct



	avg(score)
1	5

V.

3.

- I. Find the supplier who was supplied the most inventory (in dollars)
- II.  $supplies = \text{SupplierID} \bowtie f_{SUM(value)}(Purchase\_Order)$   
 $max\_supplies = f_{max(sum\_value)}(supplies)$   
 $\pi_{Name, sum(value)}(\sigma_{sum\_value = max\_supplies}(Supplier * supplies))$
- III. SELECT Name, sum(value) FROM Supplier NATURAL JOIN PurchaseOrder  
 GROUP BY SupplierID ORDER BY sum(value) DESC LIMIT 1;

IV. STATUS: Tested and Correct



```

58
59 -- Find the supplier who has supplied the most inventory
60 SELECT Name, sum(value) FROM Supplier NATURAL JOIN PurchaseOrder GROUP BY SupplierID ORDER BY sum(value) DESC LIMIT 1;
  
```

	Name	sum(value)
1	Kare	6096

V.

## Advanced Queries

1.

- I. Provide a list of member names, along with the total combined amount of all items they have rented out.
- II.  $EquipmentOrdered = (Customer \bowtie_{Customer.CustomerID=RentalOrder.CustomerID} RentalOrder)$   
 $\bowtie_{Allocation.RentalID=RentalOrder.RentalID} Allocation$

$count = \rho_{CustomerID} f_{COUNT(AllocationID)}(EquipmentOrdered)$

$\pi_{FirstName, LastName, count\_allocationid}(Customer * count)$

III. SELECT FirstName, LastName, count(AllocationID) FROM (Customer JOIN RentalOrder on Customer.CustomerID=RentalOrder.CustomerID) JOIN Allocation ON Allocation.RentalID=RentalOrder.RentalID GROUP BY Customer.CustomerID;

IV. STATUS: Tested and Correct

	FirstName	LastName	count(AllocationID)
1	Addy	Clissold	3
2	Sonny	Ramberg	1
3	Georas	Baldocci	2
4	Gare	Mullen	1
5	Carter	Wiper	1
6	Christoffer	Sharrard	4
7	Kamila	Dominici	3
8	Adelheid	Wadhams	1
9	Vanny	Batcock	2
10	Carolina	Castagnone	2
11	Lindsay	Wagenen	1
12	Glenn	Donhardt	1

V.

2.

I. Provide a list of member names and email addresses for members who have rented more equipment than the average member.

II. EquipmentOrdered = (Customer ⋈<sub>Customer.CustomerID=RentalOrder.CustomerID</sub> RentalOrder) ⋈<sub>Allocation.RentalID=RentalOrder.RentalID</sub> Allocation

$count = \rho_{count}(\rho_{CustomerID} f_{COUNT(AllocationID)}(EquipmentOrdered))$

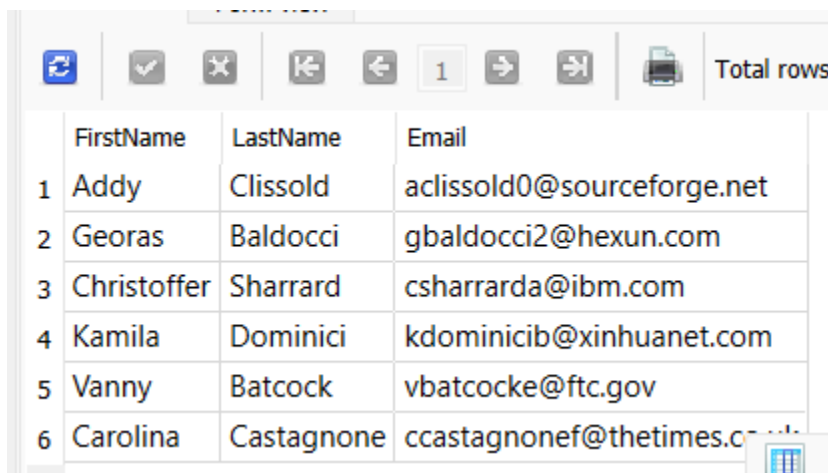
$avg = f_{avg(count)}(count)$

$\pi_{FirstName, LastName, Email}(\sigma_{count > avg}(Customer * count))$

III. 

```
SELECT FirstName, LastName, Email FROM
((Customer JOIN RentalOrder on
Customer.CustomerID=RentalOrder.CustomerID) JOIN Allocation
ON Allocation.RentalID=RentalOrder.RentalID) JOIN
(SELECT avg(count) as avg FROM (SELECT count(AllocationID) as
count
FROM (Customer JOIN RentalOrder on
Customer.CustomerID=RentalOrder.CustomerID) JOIN
Allocation ON Allocation.RentalID=RentalOrder.RentalID
GROUP BY Customer.CustomerID))
GROUP BY Customer.CustomerID
HAVING count(AllocationID) > avg;
```

IV. STATUS: Tested and Correct



	FirstName	LastName	Email
1	Addy	Clissold	aclissold0@sourceforge.net
2	Georas	Baldocci	gbaldocci2@hexun.com
3	Christoffer	Sharrard	csharrarda@ibm.com
4	Kamila	Dominici	kdominicib@xinhuanet.com
5	Vanny	Batcock	vbatcocke@ftc.gov
6	Carolina	Castagnone	ccastagnonef@thetimes.co.uk

V.

3.

- I. Provide a list of the equipment in the database and associated total copies rented to members, sorted from the equipment that has been rented the most to the equipment that has been rented the least.
- II. 
$$\text{counts} = \rho_{(\text{EquipmentSerial}, \text{count})}(\text{EquipmentSerial} \bowtie_{\text{count}(\text{EquipmentSerial})}(\text{Allocation}))$$
- III. 

```
πSerialNumber, count(Equipment ⋈SerialNumber=EquipmentSerial counts)
Select SerialNumber, count FROM Equipment LEFT OUTER JOIN (SELECT
EquipmentSerial, count(EquipmentSerial) as count FROM Allocation
GROUP BY EquipmentSerial) ON SerialNumber=EquipmentSerial ORDER
BY count DESC;
```
- IV. STATUS: Tested and Correct

	SerialNumber	count
1	90036	4
2	52297	3
3	92892	3
4	72867	2
5	84558	2
6	18823	1
7	33890	1
8	49889	1
9	55957	1
10	61913	1
11	78619	1
12	84729	1
13	94457	1
14	17988	NULL
15	23113	NULL
16	52121	NULL
17	53269	NULL
18	63907	NULL
19	70007	NULL
20	79205	NULL

V.

4.

I. Provide a list of the drones in the database and the total number of miles flown, sorted from the ones that have been delivered the highest number of items to the ones delivered the lowest.

II. deliveredOrdersDistance=

$(\text{RentalOrder} \bowtie_{\text{RentalOrder.RentalID}=\text{Delivery.RentalID}} \text{Delivery}) \bowtie_{\text{Customer.CustomerID}=\text{RentalOrder.CustomerID}} (\text{Customer} \bowtie_{\text{Customer.Warehouse}=\text{Distance.Warehouse AND Customer.Address}=\text{Distance.Address}} \text{Distance})$

$\text{droneDistances} = \rho_{(\text{DroneSerial}, \text{milesFlown})}(\text{DroneSerial} \bowtie_{\text{sum}(\text{WarehouseDistance})}(\text{deliveredOrdersDistance}))$

$\text{deliveryCounts} = \rho_{\text{SerialNumber}, \text{count}}(\text{SerialNumber} \bowtie_{\text{count}(\text{DroneSerial})}(\text{Drone} \bowtie_{\text{DroneSerial}=\text{SerialNumber}} \text{Delivery}))$

$\pi_{\text{SerialNumber}, \text{MilesFlown}}(\text{deliveryCounts} \bowtie_{\text{SerialNumber}=\text{DroneSerial}} \text{droneDistances})$

III. `SELECT SerialNumber, MilesFlown FROM (SELECT SerialNumber, count(DroneSerial) as count FROM (Drone LEFT OUTER JOIN Delivery ON DroneSerial=SerialNumber) GROUP BY SerialNumber) LEFT OUTER JOIN (SELECT DroneSerial, sum(WarehouseDistance) as MilesFlown FROM (RentalOrder JOIN Delivery ON RentalOrder.RentalID=Delivery.RentalID) JOIN (Customer JOIN Distance ON Customer.Warehouse=Distance.Warehouse AND Customer.Address=Distance.Address) ON Customer.CustomerID=RentalOrder.CustomerID GROUP BY DroneSerial) ON SerialNumber=DroneSerial ORDER BY count DESC;`

IV. STATUS: Tested and Correct

	SerialNumber	MilesFlown
1	12899	34
2	47392	42
3	57842	30
4	76289	19
5	22357	63
6	24186	19
7	28112	12
8	32043	45
9	46872	30
10	60874	1
11	71576	47
12	78684	63
13	78974	47
14	88069	7
15	94465	12
16	98716	47
17	15941	NULL
18	16702	NULL

V.

5.

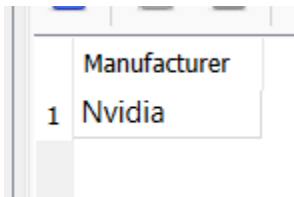
- I. Find the most popular manufacturer in the database (i.e. the one who has had the most rented items)

II.  $\text{counts} = \rho_{\text{Manufacturer}} f_{\text{COUNT(Manufacturer)}}(\text{Allocation} \bowtie_{\text{Allocation.EquipmentSerial=Inventory.SerialNumber}} \text{Inventory})$

$\pi_{\text{Manufacturer}}(f_{\text{Manufacturer}, \text{MAX(Manufacturer)}}(\text{counts}))$

III. `SELECT Manufacturer FROM Allocation JOIN Inventory ON Allocation.EquipmentSerial=Inventory.SerialNumber GROUP BY Manufacturer ORDER BY count(Manufacturer) DESC LIMIT 1;`

IV. STATUS: Tested and Correct



	Manufacturer
1	Nvidia

V.

6.

- I. Find the most used items in the database, use the running rented time of the item to calculate and provide also the number of times the item has been rented out. The output should be ordered from the highest running rented time to the lowest.

II.  $\text{equipmentCounts} = \rho_{(\text{EquipmentSerial}, \text{RentalCount})}(\text{EquipmentSerial} f_{\text{count(EquipmentSerial)}}(\text{Allocation}))$

$\text{durations} = \rho_{(\text{RentalID}, \text{Duration})}(\text{RentalOrder})$

$\text{totalDurations} = \rho_{\text{EquipmentSerial}, \text{TotalDuration}}(\text{EquipmentSerial} f_{\text{sum(Duration)}}(\text{Allocation} * \text{Duration}))$

$\pi_{\text{EquipmentSerial}, \text{TotalDuration}, \text{RentalCount}}(\text{totalDurations} * \text{equipmentCounts})$

III. `SELECT EquipmentSerial, TotalDuration, RentalCount FROM (SELECT EquipmentSerial, sum(Duration) as TotalDuration FROM Allocation NATURAL JOIN (SELECT RentalID, DueDate - CheckoutDate as Duration FROM RentalOrder) GROUP BY EquipmentSerial ORDER BY sum(Duration) DESC) NATURAL JOIN (SELECT EquipmentSerial, count(EquipmentSerial) as RentalCount FROM Allocation GROUP BY EquipmentSerial);`

IV. STATUS: Tested and Correct

	EquipmentSerial	TotalDuration	RentalCount
1	61913	1717905754519	1
2	84558	40457366000	2
3	55957	28450958000	1
4	90036	20643494000	4
5	84729	19546062000	1
6	94457	18368610000	1
7	72867	17531828000	2
8	33890	15235585000	1
9	52297	14033697000	3
10	18823	9531186000	1
11	49889	9329095000	1
12	92892	3076224000	3
13	78619	2809823000	1

V.

7.

I. Provide the names and phones of members who have rented out anything by the most demanded equipment in the database.

II.  $\text{equipmentCounts} = \rho_{(\text{EquipmentSerial}, \text{RentalID}, \text{Count})}(\text{EquipmentSerial} \bowtie \text{count}(\text{EquipmentSerial})(\text{Allocation}))$

$\text{counts} = \sigma_{\text{EquipmentSerial} \text{ NOT NULL}}(\text{Equipment} \bowtie \text{SerialNumber} = \text{EquipmentSerial} \text{ equipmentCounts})$

$\text{typeCounts} = \rho_{\text{Type}, \text{TypeCount}}(\text{Type} \bowtie \text{sum}(\text{count})(\text{counts}))$

$\pi_{\text{Firstname}, \text{LastName}, \text{PhoneNumber}}(((\text{typeCounts} * \text{Equipment}) \bowtie \text{SerialNumber} = \text{EquipmentSerial} \text{ Allocation}) \bowtie \text{RentalOrder.RentalID} = \text{RentalOrder.CustomerID} \text{ RentalOrder}) \bowtie \text{Customer.CustomerID} = \text{RentalOrder.CustomerID} \text{ Customer})$

- III. 

```
SELECT FirstName, LastName, PhoneNumber FROM
(
  SELECT Type, SUM(count) AS TypeCount FROM
  (
    SELECT EquipmentSerial, Type, count FROM
      Equipment
    LEFT OUTER JOIN
      (SELECT EquipmentSerial, RentalID,
count(EquipmentSerial) as Count FROM Allocation GROUP BY
EquipmentSerial)
      ON SerialNumber=EquipmentSerial
    WHERE EquipmentSerial NOT NULL
    GROUP BY EquipmentSerial
    ORDER BY count DESC
  )
  GROUP BY Type
  ORDER BY TypeCount DESC
LIMIT 1
) NATURAL JOIN Equipment
JOIN Allocation ON SerialNumber = EquipmentSerial
JOIN RentalOrder ON RentalOrder.RentalID = Allocation.RentalID
JOIN Customer ON Customer.CustomerID = RentalOrder.CustomerID
GROUP BY Customer.CustomerID;
```
- IV. STATUS: Tested and correct



	FirstName	LastName	PhoneNumber
1	Addy	Clissold	468-356-1143
2	Carter	Wiper	325-238-8296
3	Vanny	Batcock	638-613-2741
4	Carolina	Castagnone	841-340-3927

- V.
- 8.
- I. Provide a list of manufacturers who provided the items rented out by members who have rented more items than the average customer.
- II.  $\text{EquipmentOrdered} = (\text{Customer} \bowtie_{\text{Customer.CustomerID}=\text{RentalOrder.CustomerID}} \text{RentalOrder}) \bowtie_{\text{Allocation.RentalID}=\text{RentalOrder.RentalID}} \text{Allocation}$



$\text{count} = \rho_{\text{count}}(\text{CustomerID} \bowtie \text{COUNT}(\text{AllocationID})(\text{EquipmentOrdered}))$

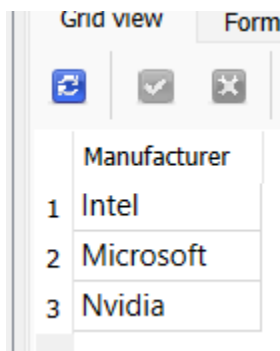
$\text{avg} = f_{\text{avg}(\text{count})}(\text{count})$

$\text{aboveAvgRenters} = (\sigma_{\text{count} > \text{avg}}(\text{Customer} * \text{count}))$

$\pi_{\text{Manufacturer}}(\text{Manufacturer} \bowtie (\text{aboveAvgRenters} \bowtie_{\text{aboveAvgRenters.CustomerID=RentalOrder.CustomerID}} \text{RentalOrder} \bowtie_{\text{Allocation.RentalID=RentalOrder.RentalID}} \text{Allocation} \bowtie_{\text{Allocation.SerialNumber=Inventory.SerialNumber}} \text{Inventory}))$

III. `SELECT Inventory.Manufacturer FROM`  
`(`  
`SELECT Customer.CustomerID FROM`  
`((Customer JOIN RentalOrder on`  
`Customer.CustomerID=RentalOrder.CustomerID) JOIN Allocation ON`  
`Allocation.RentalID=RentalOrder.RentalID)`  
`JOIN`  
`(SELECT avg(count) as avg FROM (SELECT count(AllocationID) as`  
`count`  
`FROM (Customer JOIN RentalOrder on`  
`Customer.CustomerID=RentalOrder.CustomerID) JOIN Allocation ON`  
`Allocation.RentalID=RentalOrder.RentalID`  
`GROUP BY Customer.CustomerID))`  
`GROUP BY Customer.CustomerID`  
`HAVING count(AllocationID) > avg`  
`)`  
`NATURAL JOIN RentalOrder NATURAL JOIN Allocation JOIN Inventory`  
`ON Inventory.SerialNumber=Allocation.EquipmentSerial`  
`GROUP BY Inventory.Manufacturer`

IV. STATUS: Tested and Correct



	Manufacturer	EquipmentSerial
1	Intel	
2	Microsoft	
3	Nvidia	

V.

## INSERT Samples

### Customer

```
INSERT INTO Customer (CustomerID, FirstName, LastName, Address,
PhoneNumber, Email, StartDate, Warehouse, Active) VALUES (123,
'John', 'Smith', 'Customer Street', '555-123-4567',
'exampleemail@hotmail.com', 1234567890123, '1428 Elm Street', 1)
```

There must exist a row in the Distance table with the same Address and Warehouse values.

### Allocation

```
INSERT INTO Allocation (AllocationID, EquipmentSerial, RentalID)
VALUES (12345, '123ABC', 123);
```

Requires a valid equipment serial number and rental id be created before allocation can occur

### Delivery

```
INSERT INTO Delivery (DeliveryID, DroneSerial, RentalID) VALUES
(123, '123ABC', 123);
```

Requires a valid drone serial number and rental id be created before a delivery can occur.

### Distance

```
INSERT INTO Distance (Address, Warehouse, WarehouseDistance)
VALUES ('1428 Elm Street', '123 Nearest Warehouse Street', 123);
```

Requires a valid warehouse address and customer address exists.

### Drone

```
INSERT INTO Drone (SerialNumber, Name, WeightCap, MaxSpeed,
MaxRange) VALUES ('ABC123', 'Drone Name', 123, 123, 123);
```

Requires insertion as an inventory first

### Equipment

```
INSERT INTO Equipment (SerialNumber, Weight, Description, Type,
Size) VALUES ('123ABC', 123, 'Equipment Name', 123, 123);
```

Requires insertion as an inventory first

### Inventory

```
INSERT INTO Inventory (SerialNumber, Model, Year, Status,
WarrantyExpirationDate, Manufacturer, Warehouse, DroneEquipment,
PurchaseOrder, Location) VALUES ('ABC123', 'Model Name', 2024,
'active', 1234567890123, 'Manufacturer Name', '1428 Elm Street',
0, 123, 'Location');
```

Requires valid warehouse address.

#### **Payment**

```
INSERT INTO Payment (PaymentID, Amount, Type, Date, RentalID)
VALUES (123, 123, 1, 1234567890123, 123);
```

Requires valid rental ID.

#### **PurchaseOrder**

```
INSERT INTO PurchaseOrder (SupplierID, OrderNumber, Type,
Quantity, Value, ETA, RealArrivalDate, DateMade) VALUES (123,
123, 1, 123, 123, '01/01/2024', '01/01/2024', 1234567890123)
```

Requires a valid supplier ID.

#### **RentalOrder**

```
INSERT INTO RentalOrder (RentalID, CustomerID, CheckoutDate,
DueDate, Returned, RentalFees, Description, Address) VALUES (123,
123, 1234567890123, 1234567890123, 123, 123, 'Description', '1428
Elm Street');
```

Requires a valid customer ID.

#### **Review**

```
INSERT INTO Review (RentalID, EquipmentSerial, Score) VALUES
(123, '123ABC', 'Score');
```

Requires a valid rental ID and equipment serial number.

#### **Supplier**

```
INSERT INTO Supplier (SupplierID, Name, Address) VALUES (123,
'Supplier Name', '1428 Elm Street');
```

#### **Warehouse**

```
INSERT INTO Warehouse (Address, PhoneNumber, Manager,  
StorageCapacity, DroneCapacity) VALUES ('1428 Elm Street',  
'555-123-4567', 'Manager Name', 123, 123);
```

## Delete Samples

NOTE: The database is configured with ON DELETE CASCADE for every foreign key so that referential integrity is maintained as items are deleted. With ON DELETE CASCADE, dependent rows are automatically deleted down the dependency graph.

### Customer

```
DELETE FROM Customer
WHERE CustomerID = 1
```

### Payment

```
DELETE FROM Payment
WHERE PaymentID = 10
```

### Distance

```
DELETE FROM Distance
WHERE Warehouse = '75 Carberry Road'
```

### RentalOrder

```
DELETE FROM RentalOrder
WHERE RentalID = 1
```

### Review

```
DELETE FROM Review
WHERE EquipmentSerial = 72867
```

### Delivery

```
DELETE FROM Delivery
WHERE DeliveryID = 1
```

### Allocation

```
DELETE FROM Allocation
WHERE AllocationID = 1
```

### Inventory

```
DELETE FROM Inventory
WHERE SerialNumber = 28112
```

### Drone

```
DELETE FROM Drone
WHERE SerialNumber = 41242
```

### Equipment

```
DELETE FROM Equipment
WHERE SerialNumber = 17988
```

**Warehouse**

```
DELETE FROM Warehouse
WHERE Address = "312 Bay Hill"
```

**PurchaseOrder**

```
DELETE FROM PurchaseOrder
WHERE OrderNumber = 1
```

**Supplier**

```
DELETE FROM Supplier
WHERE SupplierID = 1
```

# The SQL Database

## Section 1: Database Files

The database is available as a standalone file in our submission and included with the eclipse project. This is done so that the database used by the java application does not affect the other one and vice versa for testing purposes. The location of both files are listed in README.txt.

## Section 2: SQL scripts

The required SQL files `create.sql`, `populate.sql`, `extra-queries.sql`, `advanced-queries.sql`, and `queries.sql` are located in the under those file names within the submission as directed by `README.TXT`.



### Section 3: Program

1. The java project files for the program are zipped into app.zip, as directed in README.txt. Note that the program project contains an identical database to the standalone database.
2. Implemented Functionality
  - a. Manage Equipment
    - i. Add Equipment

```
-----
1
Please Enter Serial Number:
12345
Please Enter Description:
Test Equipment
Please Enter Weight:
11
Please Enter Type:
22
Please Enter Size:
33
Insertion Successful
-----
```

```
-----
0
-----
SerialNumber, Weight, Description, Type, Size
18823, 64, Merops sp., 5, 8
78619, 75, Acridotheres tristis, 10, 20
79205, 68, Graspus graspus, 7, 16
52121, 96, Dolichitus patagonum, 6, 24
23113, 50, Echimys chrysurus, 7, 21
53269, 27, Paraxerus cepapi, 5, 14
52297, 31, Orcinus orca, 7, 6
70007, 68, unavailable, 3, 8
49889, 92, Hyaena hyaena, 3, 5
84558, 78, Agama sp., 9, 18
92892, 50, Sagittarius serpentarius, 4, 9
17988, 62, Neophron percnopterus, 5, 18
90036, 74, Tayassu pecari, 9, 4
94457, 78, Nectarinia chalybea, 9, 8
72867, 39, Boa caninus, 7, 1
84729, 10, Ramphastos tucanus, 7, 4
55957, 35, Ploceus intermedius, 4, 9
33890, 30, Chlamydosaurus kingii, 5, 7
63907, 27, Meleagris gallinavo, 4, 8
61913, 12, Cyrtodactylus louisianensis, 2, 15
12345, 11, Test Equipment, 22, 33
-----
```

ii. Modify Equipment

-----  
Select Attribute to Change

1. Serial Number
2. Description
3. Weight
4. Type
5. Size

-----  
3

Please enter new weight

66

Edit Successful  
-----

Equipment Options

0. View Equipment
1. Add Equipment
2. Remove Equipment
3. Edit Equipment
4. Rent Equipment
5. Return Equipment
6. Deliver Equipment
7. Pickup Equipment
8. Search Equipment
9. Go Back

-----  
0  
-----

SerialNumber, Weight, Description, Type, Size

18823,	64,	Merops sp.,	5,	8
78619,	75,	Acridotheres tristis,	10,	20
79205,	68,	Graspus graspus,	7,	16
52121,	96,	Dolichitus patagonum,	6,	24
23113,	66,	Echimys chrysurus,	7,	21

iii. Remove Equipment

2

Please enter serial number of equipment to delete:

12345

Deletion Successful

0

SerialNumber,	Weight,	Description,	Type,	Size
18823,	64,	Merops sp.,	5,	8
78619,	75,	Acridotheres tristis,	10,	20
79205,	68,	Graspus graspus,	7,	16
52121,	96,	Dolichitus patagonum,	6,	24
23113,	50,	Echimys chrysurus,	7,	21
53269,	27,	Paraxerus cepapi,	5,	14
52297,	31,	Orcinus orca,	7,	6
70007,	68,	unavailable,	3,	8
49889,	92,	Hyaena hyaena,	3,	5
84558,	78,	Agama sp.,	9,	18
92892,	50,	Sagittarius serpentarius,	4,	9
17988,	62,	Neophron percnopterus,	5,	18
90036,	74,	Tayassu pecari,	9,	4
94457,	78,	Nectarinia chalybea,	9,	8
72867,	39,	Boa caninus,	7,	1
84729,	10,	Ramphastos tucanus,	7,	4
55957,	35,	Ploceus intermedius,	4,	9
33890,	30,	Chlamydosaurus kingii,	5,	7
63907,	27,	Meleagris gallopavo,	4,	8
61913,	12,	Cyrtodactylus lousiadensis,	2,	15

iv. Retrieve Equipment

8

Please enter serial number of equipment to search:

23113

-----  
SerialNumber, Weight, Description, Type, Size  
23113, 66, Echimys chrysurus, 7, 21  
-----

b. Rent Equipment

4

Please enter the serial number of equipment to be rented:

23113

Please enter the customer ID of the customer to be rented to:

12345

Please enter the date the equipment is rented:

2003-01-08

Please enter the due date for the equipment:

2005-12-06

Succesfully rented equipment  
-----

c. Return Equipment

5

Please enter the rental ID:

34568

Please enter the date the equipment returned:

2024-01-01

Successfully returned equipment  
-----

d. Deliver Equipment

```
-----  
6  
Please enter the rental ID:  
3245  
Please enter the ID of the Drone to be used:  
21455  
Please enter the date the equipment is to be delivered to:  
2003-01-08  
Successfully scheduled delivery  
-----
```

e. Pickup Equipment

```
-----  
7  
Please enter the rental ID:  
232984  
Please enter the ID of the Drone to be used:  
2138  
Please enter the date the equipment is to be picked up:  
3000-01-01  
Successfully scheduled pickup  
-----
```

- f. Reports
  - i. Renting Checkouts

```
-----
Welcome to the Database
Options:
1. Manager
2. Customer
3. Close
-----
1
-----
Welcome Manager
Options:
1. Manage Equipment:
2. Manage Drones (Not Implemented)
3. Purchase Inventory (Not Implemented)
4. View Rental Orders (Not Implemented)
5. Useful reports
6. Go Back
-----
5
-----
Select Report to Generate
1. Renting Checkouts
2. Popular Item
3. Popular Manufacturer
4. Popular Drone
5. Items Checked Out
6. Equipment by type of equipment.
7. Back
-----
1
enter a customer ID:
2
count(EquipmentSerial)
1
Bye
-----
```

ii. Popular Item

3. Close

1

Welcome Manager

Options:

1. Manage Equipment:
2. Manage Drones (Not Implemented)
3. Purchase Inventory (Not Implemented)
4. View Rental Orders (Not Implemented)
5. Useful reports
6. Go Back

5

Select Report to Generate

1. Renting Checkouts
2. Popular Item
3. Popular Manufacturer
4. Popular Drone
5. Items Checked Out
6. Equipment by type of equipment.
7. Back

2

EquipmentSerial, TotalDuration, RentalCount

61913,	1717905754519,	1
84558,	40457366000,	2
55957,	28450958000,	1
90036,	20642630000,	2
84729,	19546062000,	1
94457,	18368610000,	1
72867,	17531828000,	2
33890,	15235585000,	1
52297,	14033697000,	3
18823,	9531186000,	1
49889,	9329095000,	1
92892,	3076224000,	3
78619,	2809823000,	1

RvA

iii. Popular Manufacturer

```
The driver name is SQLite JDBC
The connection to the database was successful.
-----
Welcome to the Database
Options:
1. Manager
2. Customer
3. Close
-----
1
-----
Welcome Manager
Options:
1. Manage Equipment:
2. Manage Drones (Not Implemented)
3. Purchase Inventory (Not Implemented)
4. View Rental Orders (Not Implemented)
5. Useful reports
6. Go Back
-----
5
-----
Select Report to Generate
1. Renting Checkouts
2. Popular Item
3. Popular Manufacturer
4. Popular Drone
5. Items Checked Out
6. Equipment by type of equipment.
7. Back
-----
3
Manufacturer
Nvidia
Bye
```



iv. Popular Drone

3. Purchase Inventory (Not Implemented)
4. View Rental Orders (Not Implemented)
5. Useful reports
6. Go Back

-----  
5  
-----

Select Report to Generate

1. Renting Checkouts
2. Popular Item
3. Popular Manufacturer
4. Popular Drone
5. Items Checked Out
6. Equipment by type of equipment.
7. Back

-----  
4  
-----

SerialNumber, MilesFlown

12899,	34
47392,	42
57842,	30
76289,	19
22357,	63
24186,	19
28112,	12
32043,	45
46872,	30
60874,	1
71576,	47
78684,	63
78974,	47
88069,	7
94465,	12
98716,	47
15941,	null
16703,	null
41242,	null
87685,	null

Bye

v. Items Checked Out

```
<terminated> GRS [Java Application] /usr/lib/jvm/java-21-openj
The driver name is SQLite JDBC
The connection to the database was successful.
-----
Welcome to the Database
Options:
1. Manager
2. Customer
3. Close
-----
1
-----
Welcome Manager
Options:
1. Manage Equipment:
2. Manage Drones (Not Implemented)
3. Purchase Inventory (Not Implemented)
4. View Rental Orders (Not Implemented)
5. Useful reports
6. Go Back
-----
5
-----
Select Report to Generate
1. Renting Checkouts
2. Popular Item
3. Popular Manufacturer
4. Popular Drone
5. Items Checked Out
6. Equipment by type of equipment.
7. Back
-----
5
FirstName, LastName, count(EquipmentSerial)
Christoffer, Sharrard, 4
Bye
```

vi. Equipment by type of equipment

```
Options:
1. Manager
2. Customer
3. Close
-----
1
-----
Welcome Manager
Options:
1. Manage Equipment:
2. Manage Drones (Not Implemented)
3. Purchase Inventory (Not Implemented)
4. View Rental Orders (Not Implemented)
5. Useful reports
6. Go Back
-----
5
-----
Select Report to Generate
1. Renting Checkouts
2. Popular Item
3. Popular Manufacturer
4. Popular Drone
5. Items Checked Out
6. Equipment by type of equipment.
7. Back
-----
6
enter a year:
2017
Description
unavailable
Meleagris gallopavo
Merops sp.
Neophron percnopterus
Chlamydosaurus kingii
Echymys chrysurus
Acridotheres tristis
Bye
```