Practice Exam - Harder Version - Answer Key

2023 Final Exam Questions for [subject] . The exam duration was 2 hours; there were 17 questions, worth a total of 100 points. The questions were grouped in 5 sections.

The questions in Section A and B are all based on the following relational logical schema about a library system:

```
CREATE TABLE Book (
    ISBN VARCHAR(20) PRIMARY KEY,
    Title VARCHAR(100) NOT NULL,
    Author VARCHAR(50) NOT NULL,
    Genre VARCHAR(20) NOT NULL,
    PublicationYear INTEGER NOT NULL,
    Price INTEGER NOT NULL
);

CREATE TABLE Member (
    MemberId INTEGER PRIMARY KEY,
    Name VARCHAR(50) NOT NULL,
    Address VARCHAR(100) NOT NULL,
    PhoneNumber VARCHAR(15) NOT NULL,
    MembershipType VARCHAR(20) NOT NULL,
    JoinDate DATE NOT NULL
);

CREATE TABLE Borrowed (
    ISBN VARCHAR(20) NOT NULL,
    MemberId INTEGER NOT NULL,
    BorrowDate DATE NOT NULL,
    DueDate DATE NOT NULL,
    ReturnedDate DATE,
    PRIMARY KEY (ISBN, MemberId, BorrowDate),
    FOREIGN KEY (ISBN) REFERENCES Book(ISBN),
    FOREIGN KEY (MemberId) REFERENCES Member(MemberId)
);
```


Section A [SQL queries, 25 points in total]

Q1. [4 points] Write a SQL query to find the titles of all books that were borrowed by members who joined the library in 2022 and have a "Premium" membership type.

```
SELECT DISTINCT B.Title
FROM Book B
JOIN Borrowed BO ON B.ISBN = BO.ISBN
JOIN Member M ON BO.MemberId = M.MemberId
WHERE M.JoinDate BETWEEN '2022-01-01' AND '2022-12-31' AND M.Membership
```

Q2. [4 points] Write a SQL query to find the number of books borrowed by each member

who has borrowed at least 3 books.

```
SELECT MemberId, COUNT(*) AS TotalBooksBorrowed
FROM Borrowed
GROUP BY MemberId
HAVING COUNT(*) >= 3;
```

Q3. [5 points] Write a SQL query to find the total amount of overdue fines for all books borrowed by members who have not returned the books yet. Assume a fine of $5 per day for each day overdue.

```
SELECT SUM(5 * (JULIANDAY('now') - JULIANDAY(DueDate))) AS TotalFine
FROM Borrowed
WHERE ReturnedDate IS NULL;
```

Q4. [6 points] Write a SQL query to find the average price of books borrowed by members with a "Basic" membership type for each genre.

```
SELECT B.Genre, AVG(B.Price) AS AveragePrice
FROM Book B
JOIN Borrowed BO ON B.ISBN = BO.ISBN
JOIN Member M ON BO.MemberId = M.MemberId
WHERE M.MembershipType = 'Basic'
GROUP BY B.Genre;
```

Q5. [6 points] Write a SQL query to find the name and phone number of members who have borrowed books by the author "Stephen King" but have not returned any books yet.

```
SELECT DISTINCT M.Name, M.PhoneNumber
FROM Member M
JOIN Borrowed BO ON M.MemberId = BO.MemberId
JOIN Book B ON BO.ISBN = B.ISBN
WHERE B.Author = 'Stephen King' AND BO.ReturnedDate IS NULL;
```


Section B [Relational concepts, 25 points in total]

Q6. [5 points] In the library schema shown on page 1, the table Borrowed has a primary key which is (ISBN, MemberId, BorrowDate). Describe in English, what restrictions this puts on the contents of the table.

The primary key (ISBN, MemberId, BorrowDate) in the Borrowed table ensures that:

- **No two rows can have the same combination of ISBN, MemberId, and BorrowDate.** This means that a member cannot borrow the same book on the same date.
- **Each combination of ISBN, MemberId, and BorrowDate uniquely identifies a row in the table.** This means that we can uniquely identify a specific borrowing event using these three attributes.

Q7. [5 points] Write a relational algebra expression involving the tables in the library schema shown on page 1, whose result will answer the request “Find the titles of all books that were borrowed by members who joined the library in 2022 and have a "Premium" membership type.”

```
π Title (σ JoinDate BETWEEN '2022-01-01' AND '2022-12-31' AND Membershi
```

Q8. [7 points] Consider the SQL query below, that refers to the library schema shown on page 1:

```
SELECT MemberId, COUNT(*) AS TotalBooksBorrowed
FROM Borrowed
GROUP BY MemberId
HAVING COUNT(*) > 3;
```

Write SQL to create an index that will allow much faster execution for the query above, when the database has a large amount of data in the table. Also, explain how the index would allow the query to be calculated.

```
CREATE INDEX Borrowed_MemberId ON Borrowed (MemberId);
```

An index on the MemberId column would allow the query to be calculated much faster. The database can use this index to quickly locate all rows with the same MemberId value. Instead of scanning the entire Borrowed table, the database can use the index to efficiently retrieve the rows needed to perform the aggregation and filtering.

Q9. [8 points] Produce an ER diagram, for a conceptual model from which one could produce the relational library schema shown on page 1. Include entities, attributes, and relationships with their cardinalities.

Entities:

- Book (ISBN, Title, Author, Genre, PublicationYear, Price)
- Member (MemberId, Name, Address, PhoneNumber, MembershipType, JoinDate)
- Borrowed (ISBN, MemberId, BorrowDate, DueDate, ReturnedDate)

Relationships:

- **Borrowed:** Book (1,N) -- Member (1,N)

Section C [Conceptual model, 10 points in total]

Q10. [10 points] An online marketplace for buying and selling used electronics needs to manage its users, products, and transactions. Each user has a unique ID, name, and contact details. Products have a unique ID, description, price, and category. Transactions involve a buyer, a seller, and a product. Each transaction has a date, a quantity, and a total price. There are different types of sellers: individuals and businesses. Businesses have additional attributes like name and registered address. Draw an ER diagram of a conceptual model for the information described here, and give CREATE TABLE statements for a relational design that represents the information from your conceptual design as well as possible.

Entities:

- User (UserId, Name, ContactDetails)
- Product (ProductId, Description, Price, Category)
- Transaction (TransactionId, Date, Quantity, TotalPrice)
- Seller (SellerId, UserId, Type)
- Individual (SellerId, Name)

- Business (SellerId, Name, RegisteredAddress)

Relationships:

- **Sells:** User (1,N) -- Product (1,N)
- **Involves:** Transaction (1,1) -- Seller (1,1)
- **Involves:** Transaction (1,1) -- Buyer (1,1)
- **Involves:** Transaction (1,1) -- Product (1,1)
- **Is A:** Seller (1,1) -- Individual (1,1)
- **Is A:** Seller (1,1) -- Business (1,1)

CREATE TABLE statements:

```
CREATE TABLE User (
    UserId INTEGER PRIMARY KEY,
    Name VARCHAR(50) NOT NULL,
    ContactDetails VARCHAR(100) NOT NULL
);

CREATE TABLE Product (
    ProductId INTEGER PRIMARY KEY,
    Description VARCHAR(200) NOT NULL,
    Price INTEGER NOT NULL,
    Category VARCHAR(20) NOT NULL
);

CREATE TABLE Transaction (
    TransactionId INTEGER PRIMARY KEY,
    Date DATE NOT NULL,
    Quantity INTEGER NOT NULL,
    TotalPrice INTEGER NOT NULL,
    BuyerId INTEGER NOT NULL,
    SellerId INTEGER NOT NULL,
    ProductId INTEGER NOT NULL,
    FOREIGN KEY (BuyerId) REFERENCES User(UserId),
    FOREIGN KEY (SellerId) REFERENCES Seller(SellerId),
    FOREIGN KEY (ProductId) REFERENCES Product(ProductId)
);

CREATE TABLE Seller (
    SellerId INTEGER PRIMARY KEY,
    UserId INTEGER NOT NULL,
    Type VARCHAR(10) NOT NULL,
    FOREIGN KEY (UserId) REFERENCES User(UserId)
);

CREATE TABLE Individual (
    SellerId INTEGER PRIMARY KEY,
    Name VARCHAR(50) NOT NULL,
    FOREIGN KEY (SellerId) REFERENCES Seller(SellerId)
);

CREATE TABLE Business (
    SellerId INTEGER PRIMARY KEY,
```

```
    Name VARCHAR(50) NOT NULL,
    RegisteredAddress VARCHAR(100) NOT NULL,
    FOREIGN KEY (SellerId) REFERENCES Seller(SellerId)
);
```


The questions in Section D are all based on the following relational design, which collects data about the orders placed at an online store:

```
CREATE TABLE Orders (
    OrderID INTEGER PRIMARY KEY,
    CustomerID INTEGER NOT NULL,
    OrderDate DATE NOT NULL,
    ShippingAddress VARCHAR(100) NOT NULL,
    TotalAmount INTEGER NOT NULL
);

CREATE TABLE OrderItems (
    OrderID INTEGER NOT NULL,
    ProductID INTEGER NOT NULL,
    Quantity INTEGER NOT NULL,
    UnitPrice INTEGER NOT NULL,
    PRIMARY KEY (OrderID, ProductID),
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)
);

CREATE TABLE Product (
    ProductID INTEGER PRIMARY KEY,
    ProductName VARCHAR(50) NOT NULL,
    Category VARCHAR(20) NOT NULL,
    Price INTEGER NOT NULL
);
```

The following functional dependencies are valid in this schema:

- CustomerID ® ShippingAddress
- OrderID, ProductID ® Quantity, UnitPrice
- ProductID ® ProductName, Category, Price

Section D [Relational design theory, 25 points in total]

Q11. [4 points] Based on the Orders, OrderItems, and Product schema and functional dependencies given on page 5, explain in English the meaning of the functional dependency OrderID, ProductID ® Quantity, UnitPrice. Also, give an example of data that would not be allowed in the table OrderItems because of this dependency.

The functional dependency OrderID, ProductID ® Quantity, UnitPrice means that for a given order and a given product, the quantity and unit price must be the same for every record in the OrderItems table.

An example of data that would not be allowed:

OrderID	ProductID	Quantity	UnitPrice

1	10	2	10
1	10	3	12

This is not allowed because OrderID and ProductID are the same in both rows, but the Quantity and UnitPrice values differ.

Q12. [4 points] Based on the Orders, OrderItems, and Product schema and functional dependencies given on page 5, calculate the attribute closure (CustomerID, ProductID)+. Show the step-by-step working of the calculation. Write your answer in the box below.

1. (CustomerID, ProductID)+ = (CustomerID, ProductID)
2. (CustomerID, ProductID)+ = (CustomerID, ProductID, ShippingAddress) [using CustomerID ® ShippingAddress]
3. (CustomerID, ProductID)+ = (CustomerID, ProductID, ShippingAddress, OrderID, Quantity, UnitPrice) [using OrderID, ProductID ® Quantity, UnitPrice]
4. (CustomerID, ProductID)+ = (CustomerID, ProductID, ShippingAddress, OrderID, Quantity, UnitPrice, ProductName, Category, Price) [using ProductID ® ProductName, Category, Price]

Therefore, (CustomerID, ProductID)+ = (CustomerID, ProductID, ShippingAddress, OrderID, Quantity, UnitPrice, ProductName, Category, Price)

Q13. [5 points] Based on the Orders, OrderItems, and Product schema and functional dependencies given on page 5, the relation OrderItems is not in BCNF. Justify this statement.

The relation OrderItems is not in BCNF because the determinant (OrderID, ProductID) is not a superkey. The functional dependency OrderID, ProductID ® Quantity, UnitPrice holds, but (OrderID, ProductID) does not determine all the attributes in OrderItems. This means there is a proper subset of the determinant, which is a candidate key (OrderID).

Q14. [12 points] Based on the Orders, OrderItems, and Product schema and functional dependencies given on page 5,

- give a lossless-join decomposition of OrderItems into TWO relations;
- for each of the two decomposed relations, state the functional dependencies that hold for that relation, and state a primary key for that relation, and indicate whether or not that relation is in BCNF.
- justify that your decomposition has the lossless-join property,
- indicate whether or not your decomposition has the dependency-preserving property, and justify your decision.

Decomposition:

- **R1 (OrderID, Quantity, UnitPrice)**
- **R2 (OrderID, ProductID)**

FDs:

- **R1:** OrderID ® Quantity, UnitPrice
- **R2:** OrderID, ProductID ® OrderID, ProductID

Primary Keys:

- **R1:** OrderID
- **R2:** OrderID, ProductID

BCNF:

- **R1:** Yes, because the determinant (OrderID) is a superkey.
- **R2:** Yes, because the determinant (OrderID, ProductID) is a superkey.

Lossless-join Property:

The decomposition is lossless-join because the join of R1 and R2 on OrderID will reconstruct the original OrderItems relation. The intersection of R1 and R2 is OrderID, and OrderID is a key in both R1 and R2. Therefore, we can join them on OrderID without losing information.

Dependency-Preserving Property:

The decomposition is not dependency-preserving because the functional dependency OrderID, ProductID ® Quantity, UnitPrice is lost. The dependency is split into two separate dependencies in the decomposed relations, OrderID ® Quantity, UnitPrice and ProductID ® Quantity, UnitPrice.

Section E [Database-backed applications, 15 points in total]

Q15. [5 points] In the code you used as a skeleton for asst3 (and which had previously been used in lab for week 8), there are several places where the end-user enters some value (such as a name), and this value is then used in querying the database. Describe the aspects of the application code, which aim to prevent a SQL injection attack from damaging the data.

The code uses parameterized queries to prevent SQL injection attacks. Parameterized queries separate the SQL statement from the user-supplied data. The data is treated as a parameter and not as part of the SQL command, preventing malicious code from being injected and executed. This is done using the `execute()` method of the `psycopg2` library in Python.

Q16. [5 points] In the code you used as a skeleton for asst3 (and which had previously been used in lab for week 8), the data about end-users (such as their name and password for accessing the web application) is stored in the PostgreSQL database. Describe one security mechanism that is used in this code, to try to protect the confidentiality of the information about end-users. Discuss one attack that could be made to breach the security you described.

One security mechanism used in the code is the use of bcrypt to hash passwords before storing them in the database. Bcrypt is a strong, one-way hashing algorithm that makes it very difficult for attackers to retrieve the original password from the hashed version.

One attack that could be made to breach this security is a **dictionary attack**. An attacker could obtain a list of commonly used passwords and try to hash them using bcrypt. If the hash matches one stored in the database, the attacker would have successfully compromised the user's password.

Q17. [5 points] In the code you used as a skeleton for asst3 (and which had previously been used in lab for week 8), describe the steps by which information returned from a database query is displayed to the end-user. Mention explicitly the files and functions in the code, where each step is done.

1. **Query execution:** The database query is executed in the `database.py` file, specifically in the `execute()` method of the `Database` class.
2. **Result processing:** The result of the query, in the form of a list of tuples, is returned

from the `execute()` method to the function that called it.
3. **Data rendering:** The function that called `execute()` then processes the data and renders it into HTML, often using a templating engine like Jinja2. This step occurs in the `templates` directory, where the HTML templates are stored.
4. **Display to user:** The rendered HTML is then sent to the user's web browser for display. This is done by the `app.py` file, which handles the routing and response handling of the web application.