# E-Commerce System Database

In today's world, e-commerce has become one of the most dynamic and rapidly growing sectors in commerce. To enhance customer satisfaction and optimize business processes, it is crucial to establish a reliable and scalable database infrastructure.

The primary goal of the e-commerce system is to provide users with a seamless shopping experience and to effectively support management processes. To achieve this goal, all data transactions within the system must be conducted securely, quickly, and without errors. This is where database design comes into play.

The database is structured to securely store users' personal information, addresses, shopping carts, orders, payment information, and more. Additionally, a flexible and scalable database design is essential to support advanced features such as customer loyalty programs, multiple payment options, and personalized shopping experiences.

In the database design, various tables and relationships have been created, taking into account the needs of users and administrators. Key components such as user information, address details, product details, cart and order management, payment, and invoice information are organized through these tables. Each table is meticulously designed to ensure the consistent and related storage of relevant data.

In conclusion, this e-commerce system database design plays a critical role in enhancing the user experience and making business operations more efficient. A strong, reliable, and flexible database infrastructure will support the growth of the business and increase customer satisfaction.

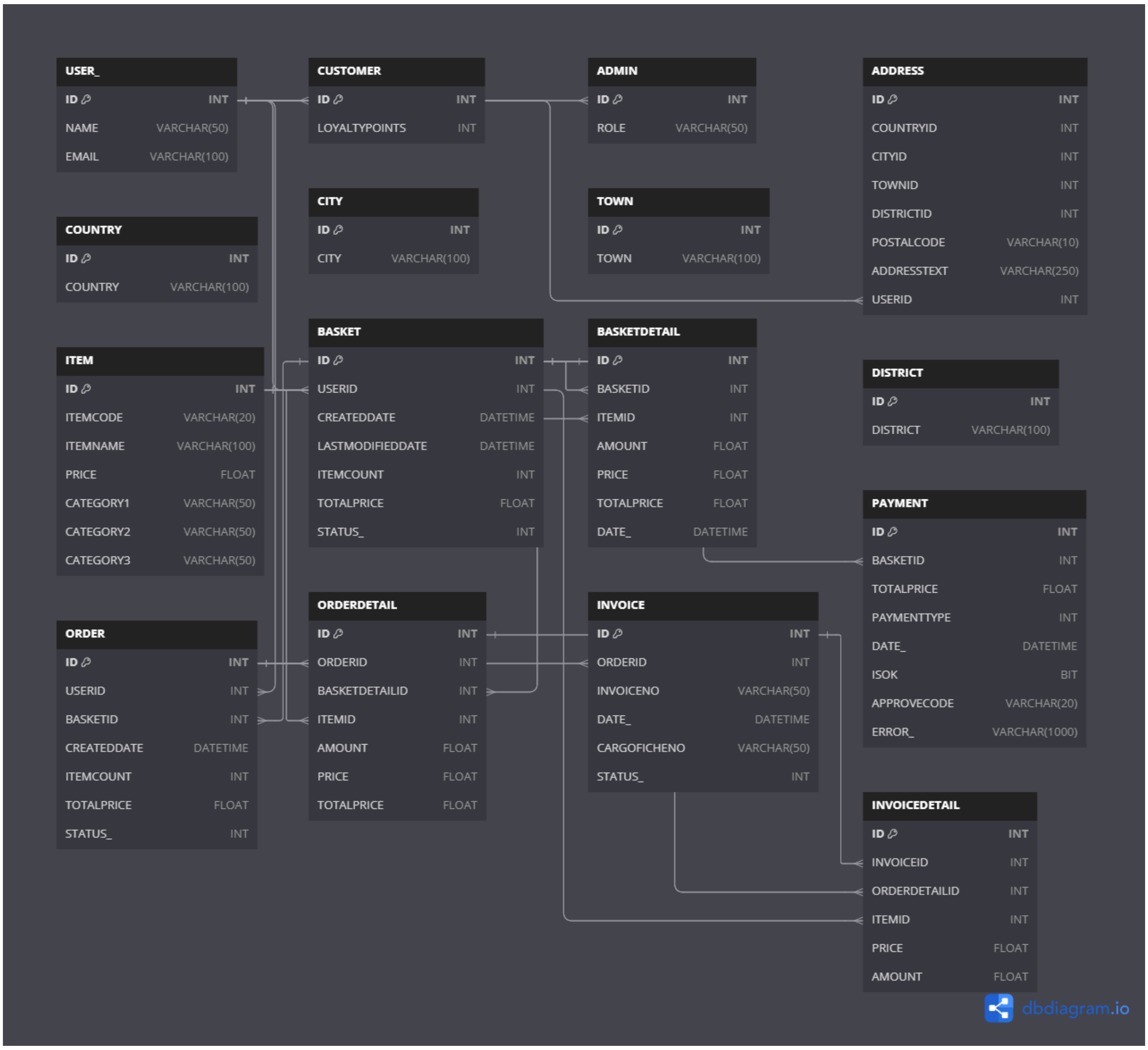
## Data Storage Requirements

1. **User Data**: User information will be stored in the USER_ table, which will include user identity information, names, and email addresses. Additionally, the CUSTOMER and ADMIN tables will differentiate between customer and administrator roles.
   - The CUSTOMER table will track customer loyalty points.
   - The ADMIN table will define the roles of administrators.
2. **Address Data**: User addresses will be stored in the ADDRESS table. This table will contain country, city, town, district, postal code, and detailed address information and will be linked to the USER_ table.
3. **Location Data**: The COUNTRY, CITY, TOWN, and DISTRICT tables will store country, city, town, and district information, respectively. These tables will be linked to the ADDRESS table and will ensure hierarchical organization of address information.

4. **Product Information**: Detailed information about products will be stored in the ITEM table. This table will include product code, product name, price, and category information.
5. **Cart and Order Management**: User shopping carts and orders will be managed through the BASKET, BASKETDETAIL, ORDER, and ORDERDETAIL tables.
    - The BASKET table will store information related to the user's cart, including creation and last modification dates, item count, total price, and cart status.
    - The BASKETDETAIL table will contain detailed information about the items in the cart.
    - The ORDER table will hold general information about user orders.
    - The ORDERDETAIL table will store detailed information about the contents of the orders.
6. **Payment Information**: The PAYMENT table will store information related to user payment transactions. This table will include payment type, total amount, approval code, and error messages.
7. **Invoice Management**: The INVOICE and INVOICEDETAIL tables will be used to manage invoice information related to user orders.
    - The INVOICE table will contain general invoice information related to an order.
    - The INVOICEDETAIL table will store detailed information about the items within the invoice.

This database design aims to meet the requirements of the e-commerce system and provide a structure that fulfills the expectations of users.

## E-R diagram



This database schema includes a data model designed for e-commerce platforms. The schema consists of tables for users, customers, admins, addresses, countries, cities, towns, districts, items, baskets, payments, orders, invoices, and their respective details.

The **USER_** table stores basic information for all users, while the **CUSTOMER** and **ADMIN** tables provide further details based on user types. The **CUSTOMER** table includes loyalty points for customers, and the **ADMIN** table specifies roles for administrators.

The **ADDRESS** table details users' address information, supported by the **COUNTRY**, **CITY**, **TOWN**, and **DISTRICT** tables that maintain a hierarchical structure for addresses.

The **ITEM** table stores product information such as code, name, price, and categories. The **BASKET** and **BASKETDETAIL** tables track the items users add to their baskets. The **BASKET** table holds general basket information, while the **BASKETDETAIL** table provides detailed content of the baskets.

The **PAYMENT** table contains payment-related information, including payment type, total price, date, and approval code. The **ORDER** and **ORDERDETAIL** tables cover order information and details. The **ORDER** table stores general order information, and the **ORDERDETAIL** table details the contents of the orders.

The **INVOICE** and **INVOICEDETAIL** tables support billing processes. The **INVOICE** table stores general invoice information, while the **INVOICEDETAIL** table contains detailed invoice information.

# Functional Dependencies in the Database

Here are the identified functional dependencies (FDs) in the database, outlining how attributes depend on one another.

Table: USER_

1. ID → NAME, EMAIL

Table: CUSTOMER

1. ID → LOYALTYPOINTS

Table: ADMIN

1. ID → ROLE

Table: ADDRESS

1. ID → COUNTRYID, CITYID, TOWNID, DISTRICTID, POSTALCODE, ADDRESSTEXT, USERID
2. USERID → COUNTRYID, CITYID, TOWNID, DISTRICTID, POSTALCODE, ADDRESSTEXT (assuming a user can have only one address)

Table: COUNTRY

1. ID → COUNTRY

Table: CITY

1. ID → CITY

Table: TOWN

1. ID → TOWN

Table: DISTRICT

1. ID → DISTRICT

Table: ITEM

1. ID → ITEMCODE, ITEMNAME, PRICE, CATEGORY1, CATEGORY2, CATEGORY3
2. ITEMCODE → ITEMNAME, PRICE, CATEGORY1, CATEGORY2, CATEGORY3

Table: BASKET

1. ID → USERID, CREATEDDATE, LASTMODIFIEDDATE, ITEMCOUNT, TOTALPRICE, STATUS_

Table: BASKETDETAIL

1. ID → BASKETID, ITEMID, AMOUNT, PRICE, TOTALPRICE, DATE_
2. BASKETID, ITEMID → AMOUNT, PRICE, TOTALPRICE, DATE_

Table: PAYMENT

1. ID → BASKETID, TOTALPRICE, PAYMENTTYPE, DATE_, ISOK, APPROVECODE, ERROR_

Table: ORDER

1. ID → USERID, BASKETID, CREATEDDATE, ITEMCOUNT, TOTALPRICE, STATUS_

Table: ORDERDETAIL

1. ID → ORDERID, BASKETDETAILID, ITEMID, AMOUNT, PRICE, TOTALPRICE

Table: INVOICE

1. ID → ORDERID, INVOICENO, DATE_, CARGOFICHENO, STATUS_
2. ORDERID → INVOICENO, DATE_, CARGOFICHENO, STATUS_

Table: INVOICEDETAIL

1. ID → INVOICEID, ORDERDETAILID, ITEMID, PRICE, AMOUNT

## Normalization

The process of normalizing tables is to organize them according to specific rules to reduce redundancy and inconsistency in the database. These rules are called First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), and Boyce-Codd Normal Form (BCNF). Below, I will define each normal form and demonstrate how the tables comply with these normal forms.

### 1. First Normal Form (1NF) Definition:

- Each table cell contains atomic (indivisible) values.
- Each row of data is unique.
- Each column contains the same type of data.

All the tables are already in 1NF because each cell contains atomic values, and each column contains the same type of data. For example, the USER_ table has columns like ID, NAME, and EMAIL that contain atomic values.

### 2. Second Normal Form (2NF) Definition:

- Must be in 1NF.
- All attributes (columns) must depend on the entire primary key (no partial dependency).

If a table has a primary key composed of a single column, it is already in 2NF. For instance, let's consider tables like BASKETDETAIL and ORDERDETAIL. For example:

```
CREATE TABLE BASKETDETAIL (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    BASKETID INT,
    ITEMID INT,
    AMOUNT FLOAT,
    PRICE FLOAT,
    TOTALPRICE FLOAT,
    DATE_ DATETIME,
    FOREIGN KEY (BASKETID) REFERENCES BASKET(ID),
    FOREIGN KEY (ITEMID) REFERENCES ITEM(ID)
);
```

In this table, ID is the primary key, and all other columns depend on ID, meaning there is no partial dependency.

### 3. Third Normal Form (3NF) Definition:

- Must be in 2NF.

- No attribute should depend on another non-primary key attribute (no transitive dependency).

For example, consider the USER_ and CUSTOMER tables:

```
CREATE TABLE CUSTOMER (
    ID INT PRIMARY KEY,
    LOYALTYPOINTS INT,
    FOREIGN KEY (ID) REFERENCES USER_(ID)
);
```

This table is in 2NF, and LOYALTYPOINTS depends on the ID column. Since there is no other dependency, it is in 3NF.

## 4. Boyce-Codd Normal Form (BCNF) Definition:

- Must be in 3NF.
- Every determinant must be a super key.

If a table is in 3NF and every determinant (attribute on which other attributes depend) is a super key, the table is in BCNF. For example:

```
CREATE TABLE BASKET (
    ID INT AUTO_INCREMENT PRIMARY KEY,
    USERID INT,
    CREATEDDATE DATETIME,
    LASTMODIFIEDDATE DATETIME,
    ITEMCOUNT INT,
    TOTALPRICE FLOAT,
    STATUS_ INT,
    FOREIGN KEY (USERID) REFERENCES USER_(ID)
);
```

In this table, ID is the super key, and all other attributes depend on ID. Therefore, the table is in BCNF.

These explanations illustrate how normalization, weak entities, and unique entities are applied in database design and how they differ from each other.

## Database Implementation

In this database design, various relationships between tables are defined to ensure data integrity and efficient management. The USER_ table contains basic user information, while the CUSTOMER and ADMIN tables are related to the USER_ table. The CUSTOMER table's ID field references the ID field in the USER_ table, ensuring that each customer record is linked to a user record. Similarly, the ADMIN table's ID field references the USER_ table, ensuring that each admin record is linked to a user record.

The ADDRESS table stores user address information and references the USER_ table's ID field, linking each address record to a user. Additionally, the ADDRESS table's COUNTRYID, CITYID, TOWNID, and DISTRICTID fields reference the ID fields in the COUNTRY, CITY, TOWN, and DISTRICT tables, respectively, allowing detailed tracking of address components.

The BASKET table contains users' shopping cart information and references the USER_ table's ID field. The BASKETDETAIL table stores details of each basket and references the BASKET table's ID field. It also references the ITEM table's ID field, linking each basket detail to product information.

The ORDER table contains user order information and references the ID fields in both the USER_ and BASKET tables. The ORDERDETAIL table stores details of each order and references the ID fields in the ORDER and BASKETDETAIL tables. It also references the ITEM table's ID field, linking each order detail to product information.

The INVOICE table contains invoice information for orders and references the ORDER table's ID field. The INVOICEDETAIL table stores details of each invoice and references the ID fields in both the INVOICE and ORDERDETAIL tables. It also references the ITEM table's ID field, linking each invoice detail to product information.

Finally, the PAYMENT table stores payment information and references the BASKET table's ID field, ensuring that each payment record is linked to a shopping cart. These relationships maintain data integrity and consistency, providing effective management of the database.

## Keys

### USER_ Table

- **Primary Key (PK):** `ID`

### CUSTOMER Table

- **Primary Key (PK):** `ID`
- **Foreign Key (FK):** `ID` references `USER_.ID`

### ADMIN Table

- **Primary Key (PK):** `ID`
- **Foreign Key (FK):** `ID` references `USER_.ID`

### ADDRESS Table

- **Primary Key (PK):** `ID`
- **Foreign Key (FK):** `USERID` references `USER_.ID`
- **Foreign Key (FK):** `COUNTRYID` references `COUNTRY.ID`
- **Foreign Key (FK):** `CITYID` references `CITY.ID`
- **Foreign Key (FK):** `TOWNID` references `TOWN.ID`
- **Foreign Key (FK):** `DISTRICTID` references `DISTRICT.ID`

### COUNTRY Table

- **Primary Key (PK):** `ID`

### CITY Table

- **Primary Key (PK):** `ID`

### TOWN Table

- **Primary Key (PK):** `ID`

### DISTRICT Table

- **Primary Key (PK):** `ID`

### ITEM Table

- **Primary Key (PK):** `ID`

### BASKET Table

- **Primary Key (PK):** `ID`
- **Foreign Key (FK):** `USERID` references `USER_.ID`

### BASKETDETAIL Table

- **Primary Key (PK):** `ID`
- **Foreign Key (FK):** `BASKETID` references `BASKET.ID`
- **Foreign Key (FK):** `ITEMID` references `ITEM.ID`

### PAYMENT Table

- **Primary Key (PK):** `ID`
- **Foreign Key (FK):** `BASKETID` references `BASKET.ID`

### ORDER Table

- **Primary Key (PK):** `ID`
- **Foreign Key (FK):** `USERID` references `USER_.ID`
- **Foreign Key (FK):** `BASKETID` references `BASKET.ID`

### ORDERDETAIL Table

- **Primary Key (PK):** `ID`
- **Foreign Key (FK):** `ORDERID` references `ORDER.ID`
- **Foreign Key (FK):** `BASKETDETAILID` references `BASKETDETAIL.ID`
- **Foreign Key (FK):** `ITEMID` references `ITEM.ID`

### INVOICE Table

- **Primary Key (PK):** `ID`
- **Foreign Key (FK):** `ORDERID` references `ORDER.ID`

### INVOICEDETAIL Table

- **Primary Key (PK):** `ID`
- **Foreign Key (FK):** `INVOICEID` references `INVOICE.ID`
- **Foreign Key (FK):** `ORDERDETAILID` references `ORDERDETAIL.ID`
- **Foreign Key (FK):** `ITEMID` references `ITEM.ID`

## TABLES

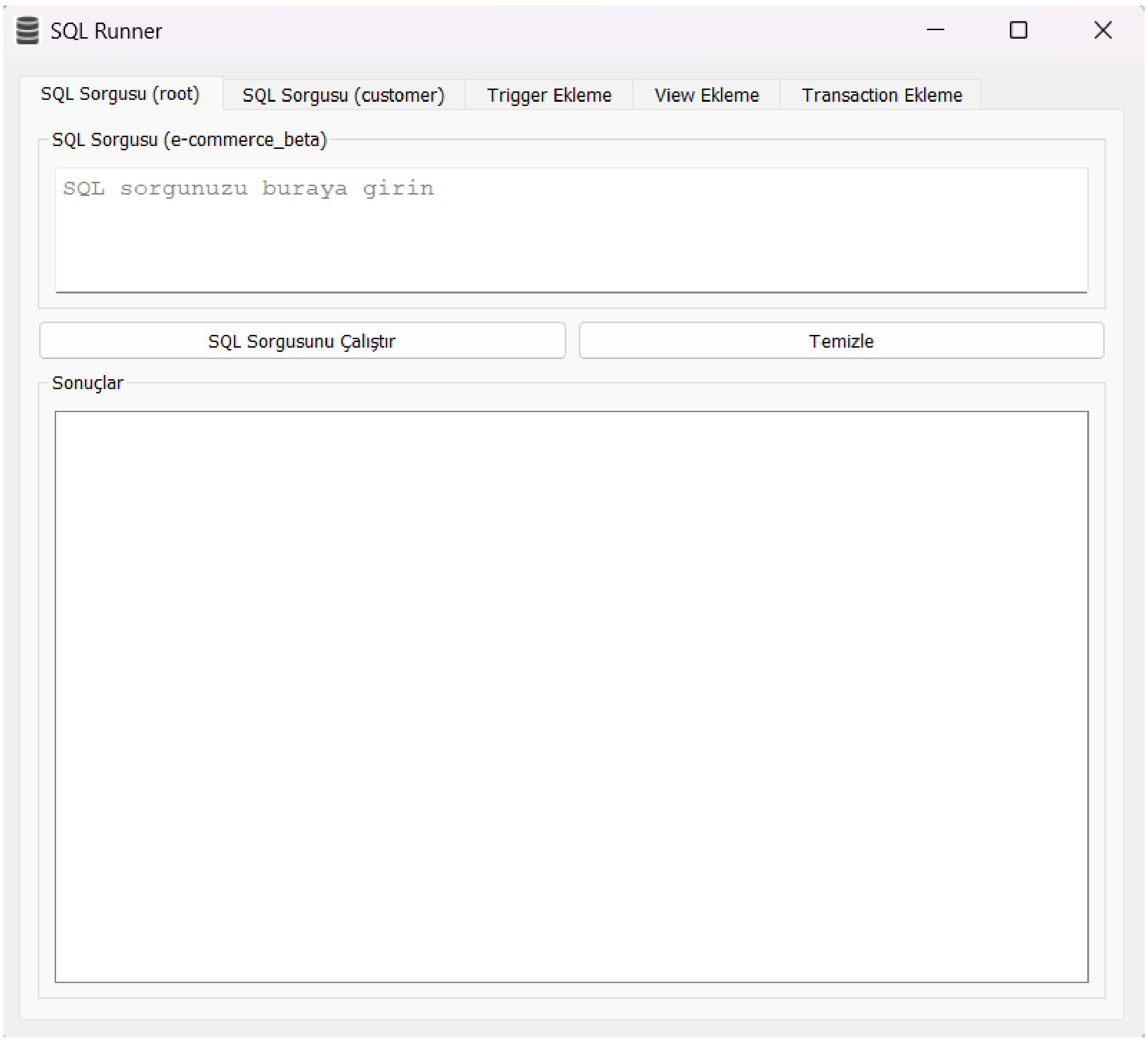
| | | | |
|---|---|---|---|
| USER_ | CUSTOMER | ADMIN | ADDRESS |
| COUNTRY | CITY | TOWN | DISTRICT |
| ITEM | BASKET | BASKETDETAIL | PAYMENT |
| ORDER | ORDERDETAIL | INVOICE | INVOICEDETAIL |

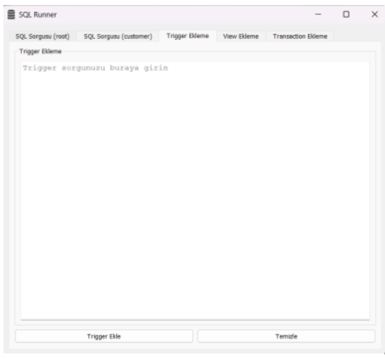
- **USER_**: Stores information about all users in the system, including both customers and administrators.
- **CUSTOMER**: Contains customer-specific information and loyalty points.
- **ADMIN**: Contains administrator-specific information and roles.
- **ADDRESS**: Stores address details of users, including country, city, town, district, postal code, and address text.
- **COUNTRY**: Stores information about countries.
- **CITY**: Stores information about cities.
- **TOWN**: Stores information about towns.
- **DISTRICT**: Stores information about districts.
- **ITEM**: Stores product information, including item code, name, price, and categories.
- **BASKET**: Stores information about user shopping baskets, including creation date, item count, total price, and status.
- **BASKETDETAIL**: Contains detailed information about items in a basket, including quantity, price, and total price.
- **PAYMENT**: Stores payment details, including total price, payment type, approval code, and error message.
- **ORDER**: Contains information about user orders, including creation date, item count, total price, and status.
- **ORDERDETAIL**: Stores detailed information about order items, including quantity, price, and total price.
- **INVOICE**: Stores invoice information, including invoice number, date, and cargo receipt number.
- **INVOICEDETAIL**: Contains detailed information about items in an invoice, including quantity and price.

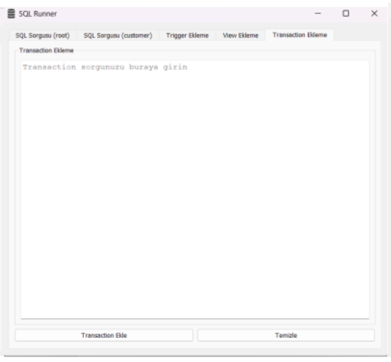
**Weak Entity:** • A weak entity is one that depends on another entity for its existence and typically uses part of the primary key of another entity as part of its own key. In this schema, the ADDRESS table can be considered a weak entity because it uses a composite primary key consisting of USERID and ID, indicating a dependent relationship on USERID.

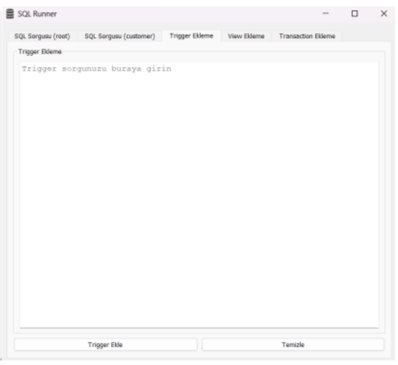
**Unique Entity:** • A unique entity is defined uniquely by its primary key. In this schema, all tables use primary keys (ID columns), and some columns are also defined as unique. For example: o In the USER_ table, the USERNAME_, EMAIL, and TELNR1 columns are unique. o In the COUNTRY, TOWN, DISTRICT, and CITY tables, the name columns are unique. o In the ITEM table, the ITEMCODE is unique. o In the INVOICE table, the INVOICENO is unique. These unique columns ensure the uniqueness of these entities.

## User Interface









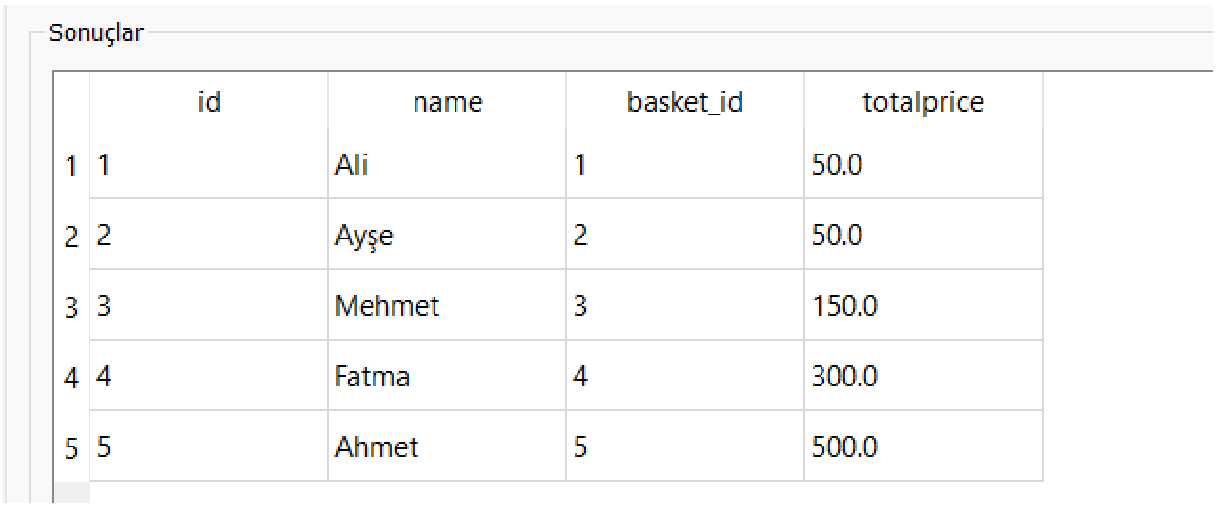
This image shows an SQL query interface written using PyQt. This interface allows users to execute SQL queries and view the results. At the top, there are tabs for different SQL users (root, customer) and various addition operations (trigger, view, transaction).

# Query Development

- **INNER JOIN**

```
SELECT
    user_.id,
    user_.name,
    basket.id AS basket_id,
    basket.totalprice
FROM
    user_
INNER JOIN
    basket
ON
    user_.id = basket.userid;
```

This query retrieves records that match in the user_ and basket tables based on the userid column.



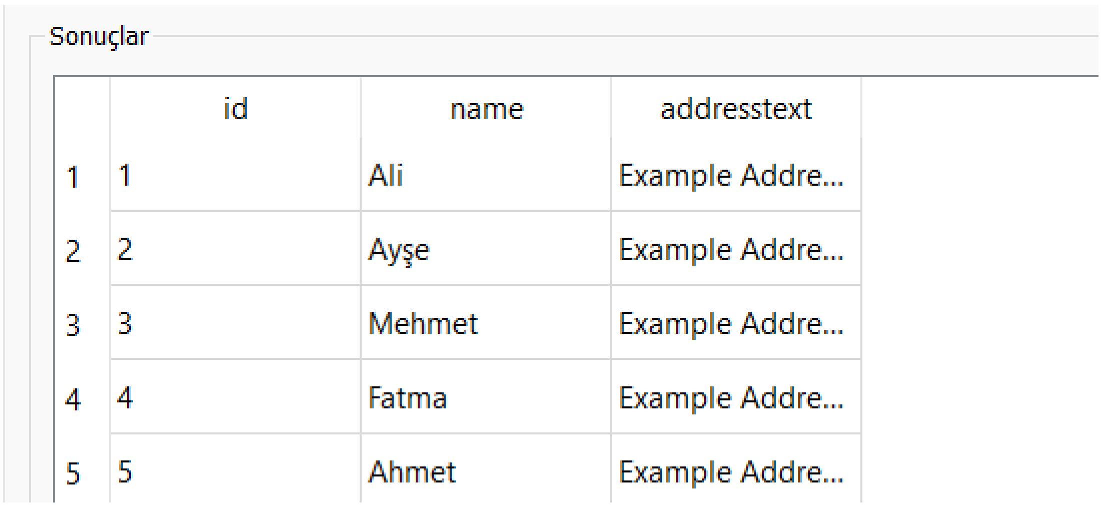
Sonuçlar

| | id | name | basket_id | totalprice |
|---|---|---|---|---|
| 1 | 1 | Ali | 1 | 50.0 |
| 2 | 2 | Ayşe | 2 | 50.0 |
| 3 | 3 | Mehmet | 3 | 150.0 |
| 4 | 4 | Fatma | 4 | 300.0 |
| 5 | 5 | Ahmet | 5 | 500.0 |

- **LEFT JOIN**

```
SELECT
    user_.id,
    user_.name,
    address.addresstext
FROM
    user_
LEFT JOIN
    address
ON
    user_.id = address.userid;
```

This query retrieves all records from the user_ table and the matching records from the address table based on the userid column. For records in the user_ table that do not have a match in the address table, the addresstext will be NULL.

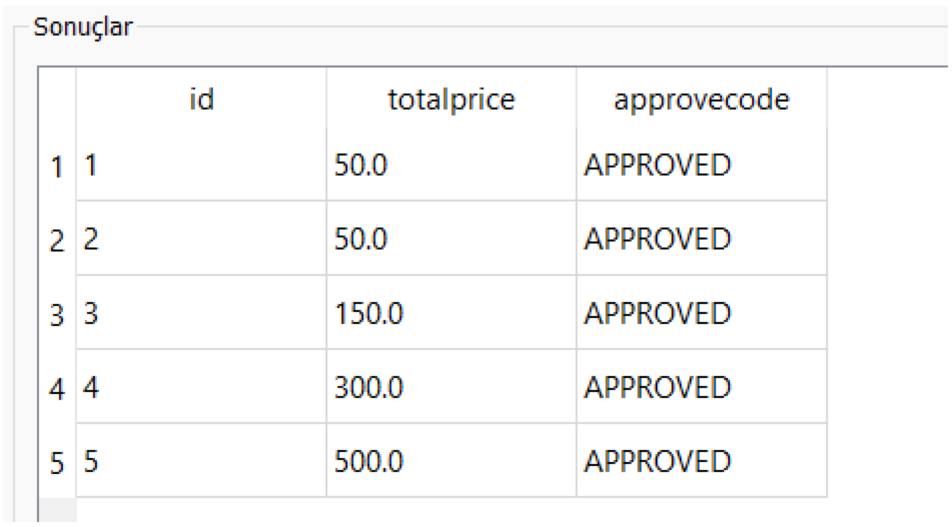

Sonuçlar

| | id | name | addresstext |
|---|---|---|---|
| 1 | 1 | Ali | Example Addre... |
| 2 | 2 | Ayşe | Example Addre... |
| 3 | 3 | Mehmet | Example Addre... |
| 4 | 4 | Fatma | Example Addre... |
| 5 | 5 | Ahmet | Example Addre... |

- **RIGHT JOIN**

```
SELECT
    orders.id,
    orders.totalprice,
    payment.approvecode
FROM
    orders
RIGHT JOIN
    payment
ON
    orders.id = payment.basketid;
```

This query retrieves all records from the payment table and the matching records from the orders table based on the basketid column. For records in the payment table that do not have a match in the orders table, the totalprice will be NULL.


Sonuçlar

| | id | totalprice | approvecode |
|---|---|---|---|
| 1 | 1 | 50.0 | APPROVED |
| 2 | 2 | 50.0 | APPROVED |
| 3 | 3 | 150.0 | APPROVED |
| 4 | 4 | 300.0 | APPROVED |
| 5 | 5 | 500.0 | APPROVED |

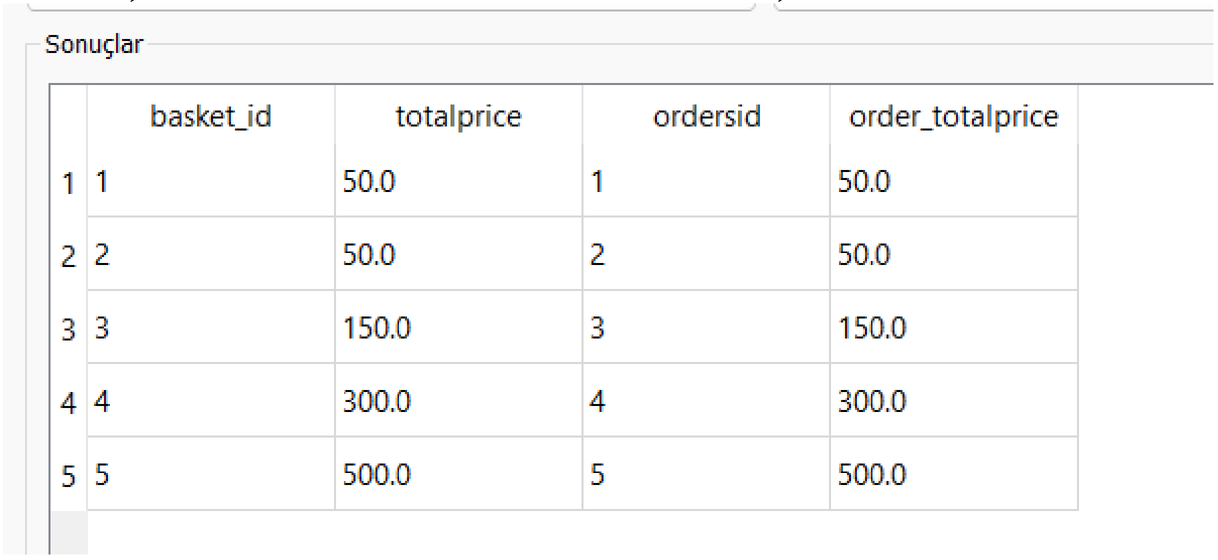
- **FULL OUTER JOIN**

```
SELECT
    basket.id AS basket_id,
    basket.totalprice,
    orders.id AS ordersid,
    orders.totalprice AS order_totalprice
FROM
    basket
LEFT JOIN
    orders
ON
    basket.userid = orders.userid

UNION

SELECT
    basket.id AS basket_id,
    basket.totalprice,
    orders.id AS ordersid,
    orders.totalprice AS orderstotalprice
FROM
    orders
RIGHT JOIN
    basket
ON
    orders.userid = basket.userid;
```

This query retrieves all records from both the basket and orders tables. Matching records are combined, and for records without a match in the other table, NULL values are returned.



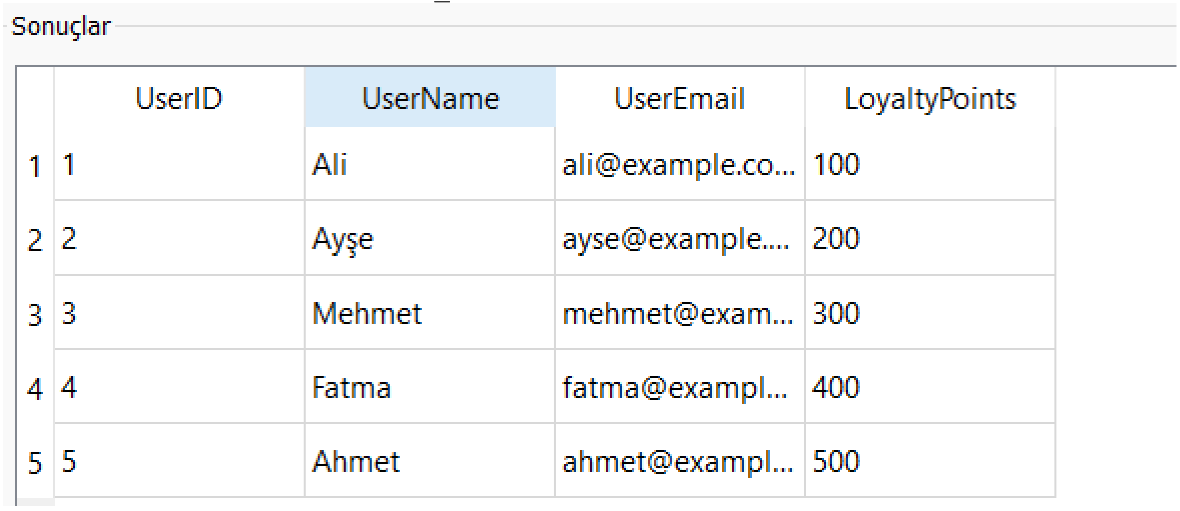
Sonuçlar

| | basket_id | totalprice | ordersid | order_totalprice |
|---|---|---|---|---|
| 1 | 1 | 50.0 | 1 | 50.0 |
| 2 | 2 | 50.0 | 2 | 50.0 |
| 3 | 3 | 150.0 | 3 | 150.0 |
| 4 | 4 | 300.0 | 4 | 300.0 |
| 5 | 5 | 500.0 | 5 | 500.0 |

## Views

- This view provides a list of users and their associated loyalty points as customers.

```
CREATE VIEW v1 AS
SELECT
    USER_.ID AS UserID,
    USER_.NAME AS UserName,
    USER_.EMAIL AS UserEmail,
    CUSTOMER.LOYALTYPOINTS AS LoyaltyPoints
FROM USER_
JOIN CUSTOMER ON USER_.ID = CUSTOMER.ID;
```

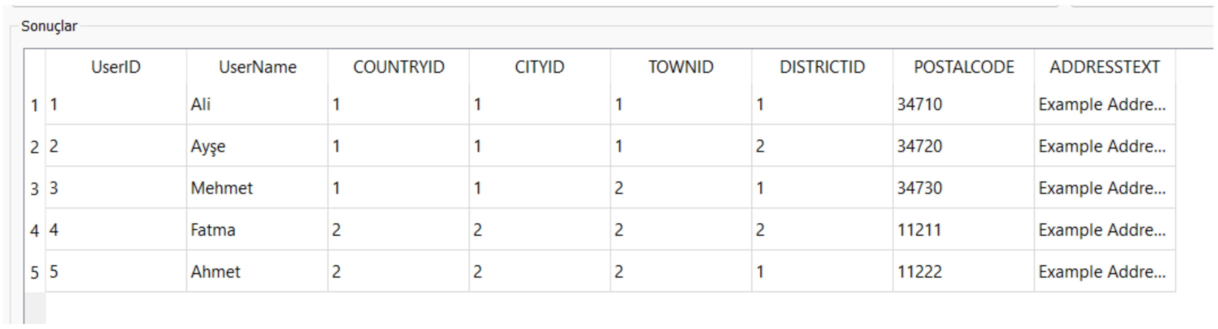


Sonuçlar

| | UserID | UserName | UserEmail | LoyaltyPoints |
|---|---|---|---|---|
| 1 | 1 | Ali | ali@example.co... | 100 |
| 2 | 2 | Ayşe | ayse@example.... | 200 |
| 3 | 3 | Mehmet | mehmet@exam... | 300 |
| 4 | 4 | Fatma | fatma@exampl... | 400 |
| 5 | 5 | Ahmet | ahmet@exampl... | 500 |

- This view lists users and their address information.

```
CREATE VIEW v2 AS
SELECT
    USER_.ID AS UserID,
    USER_.NAME AS UserName,
    ADDRESS.COUNTRYID,
    ADDRESS.CITYID,
    ADDRESS.TOWNID,
    ADDRESS.DISTRICTID,
    ADDRESS.POSTALCODE,
    ADDRESS.ADDRESSTEXT
FROM USER_
JOIN ADDRESS ON USER_.ID = ADDRESS.USERID;
```

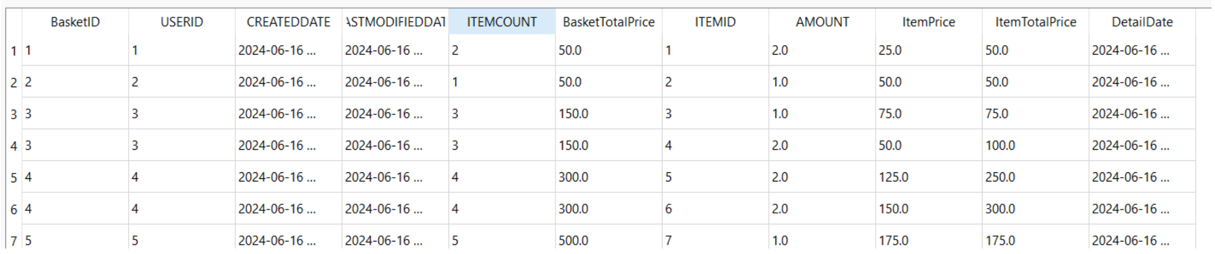


Sonuçlar

| | UserID | UserName | COUNTRYID | CITYID | TOWNID | DISTRICTID | POSTALCODE | ADDRESSTEXT |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Ali | 1 | 1 | 1 | 1 | 34710 | Example Addre... |
| 2 | 2 | Ayşe | 1 | 1 | 1 | 2 | 34720 | Example Addre... |
| 3 | 3 | Mehmet | 1 | 1 | 2 | 1 | 34730 | Example Addre... |
| 4 | 4 | Fatma | 2 | 2 | 2 | 2 | 11211 | Example Addre... |
| 5 | 5 | Ahmet | 2 | 2 | 2 | 1 | 11222 | Example Addre... |

- This view provides information about baskets and their related basket details.

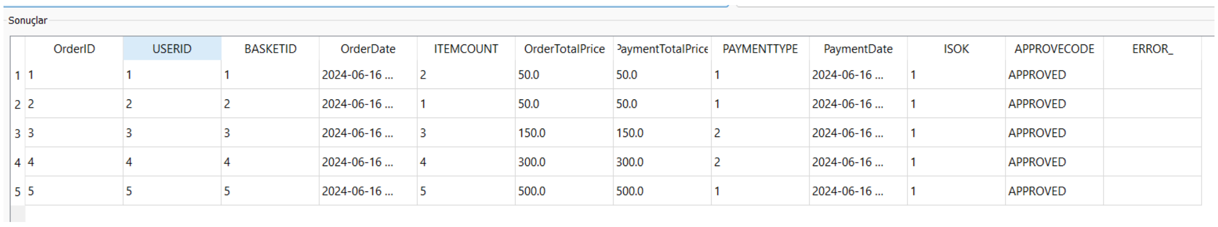
```
CREATE VIEW v3 AS
SELECT
    BASKET.ID AS BasketID,
    BASKET.USERID,
    BASKET.CREATEDDATE,
    BASKET.LASTMODIFIEDDATE,
    BASKET.ITEMCOUNT,
    BASKET.TOTALPRICE AS BasketTotalPrice,
    BASKETDETAIL.ITEMID,
    BASKETDETAIL.AMOUNT,
    BASKETDETAIL.PRICE AS ItemPrice,
    BASKETDETAIL.TOTALPRICE AS ItemTotalPrice,
    BASKETDETAIL.DATE_ AS DetailDate
FROM BASKET
JOIN BASKETDETAIL ON BASKET.ID = BASKETDETAIL.BASKETID;
```



| | BasketID | USERID | CREATEDDATE | \STMODIFIEDDAT | ITEMCOUNT | BasketTotalPrice | ITEMID | AMOUNT | ItemPrice | ItemTotalPrice | DetailDate |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2024-06-16 ... | 2024-06-16 ... | 2 | 50.0 | 1 | 2.0 | 25.0 | 50.0 | 2024-06-16 ... |
| 2 | 2 | 2 | 2024-06-16 ... | 2024-06-16 ... | 1 | 50.0 | 2 | 1.0 | 50.0 | 50.0 | 2024-06-16 ... |
| 3 | 3 | 3 | 2024-06-16 ... | 2024-06-16 ... | 3 | 150.0 | 3 | 1.0 | 75.0 | 75.0 | 2024-06-16 ... |
| 4 | 3 | 3 | 2024-06-16 ... | 2024-06-16 ... | 3 | 150.0 | 4 | 2.0 | 50.0 | 100.0 | 2024-06-16 ... |
| 5 | 4 | 4 | 2024-06-16 ... | 2024-06-16 ... | 4 | 300.0 | 5 | 2.0 | 125.0 | 250.0 | 2024-06-16 ... |
| 6 | 4 | 4 | 2024-06-16 ... | 2024-06-16 ... | 4 | 300.0 | 6 | 2.0 | 150.0 | 300.0 | 2024-06-16 ... |
| 7 | 5 | 5 | 2024-06-16 ... | 2024-06-16 ... | 5 | 500.0 | 7 | 1.0 | 175.0 | 175.0 | 2024-06-16 ... |

- This view provides a list of orders and their associated payment information.

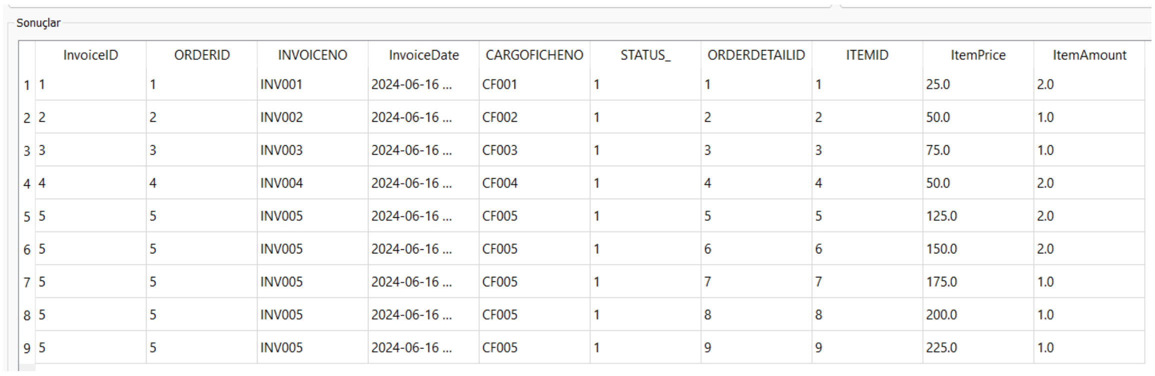
```
CREATE VIEW v4 AS
SELECT
    `ORDERS`.ID AS OrderID,
    `ORDERS`.USERID,
    `ORDERS`.BASKETID,
    `ORDERS`.CREATEDDATE AS OrderDate,
    `ORDERS`.ITEMCOUNT,
    `ORDERS`.TOTALPRICE AS OrderTotalPrice,
    PAYMENT.TOTALPRICE AS PaymentTotalPrice,
    PAYMENT.PAYMENTTYPE,
    PAYMENT.DATE_ AS PaymentDate,
    PAYMENT.ISOK,
    PAYMENT.APPROVECODE,
    PAYMENT.ERROR_
FROM `ORDERS`
JOIN PAYMENT ON `ORDERS`.BASKETID = PAYMENT.BASKETID;
```


Sonuçlar

| | OrderID | USERID | BASKETID | OrderDate | ITEMCOUNT | OrderTotalPrice | aymentTotalPrice | PAYMENTTYPE | PaymentDate | ISOK | APPROVECODE | ERROR_ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 2024-06-16 ... | 2 | 50.0 | 50.0 | 1 | 2024-06-16 ... | 1 | APPROVED | |
| 2 | 2 | 2 | 2 | 2024-06-16 ... | 1 | 50.0 | 50.0 | 1 | 2024-06-16 ... | 1 | APPROVED | |
| 3 | 3 | 3 | 3 | 2024-06-16 ... | 3 | 150.0 | 150.0 | 2 | 2024-06-16 ... | 1 | APPROVED | |
| 4 | 4 | 4 | 4 | 2024-06-16 ... | 4 | 300.0 | 300.0 | 2 | 2024-06-16 ... | 1 | APPROVED | |
| 5 | 5 | 5 | 5 | 2024-06-16 ... | 5 | 500.0 | 500.0 | 1 | 2024-06-16 ... | 1 | APPROVED | |

- This view lists invoices and their related invoice details.

```
CREATE VIEW v5 AS
SELECT
    INVOICE.ID AS InvoiceID,
    INVOICE.ORDERID,
    INVOICE.INVOICENO,
    INVOICE.DATE_ AS InvoiceDate,
    INVOICE.CARGOFICHENO,
    INVOICE.STATUS_,
    INVOICEDETAIL.ORDERDETAILID,
    INVOICEDETAIL.ITEMID,
    INVOICEDETAIL.PRICE AS ItemPrice,
    INVOICEDETAIL.AMOUNT AS ItemAmount
FROM INVOICE
JOIN INVOICEDETAIL ON INVOICE.ID = INVOICEDETAIL.INVOICEID;
```


Sonuçlar

| | InvoiceID | ORDERID | INVOICENO | InvoiceDate | CARGOFICHENO | STATUS_ | ORDERDETAILID | ITEMID | ItemPrice | ItemAmount |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | INV001 | 2024-06-16 ... | CF001 | 1 | 1 | 1 | 25.0 | 2.0 |
| 2 | 2 | 2 | INV002 | 2024-06-16 ... | CF002 | 1 | 2 | 2 | 50.0 | 1.0 |
| 3 | 3 | 3 | INV003 | 2024-06-16 ... | CF003 | 1 | 3 | 3 | 75.0 | 1.0 |
| 4 | 4 | 4 | INV004 | 2024-06-16 ... | CF004 | 1 | 4 | 4 | 50.0 | 2.0 |
| 5 | 5 | 5 | INV005 | 2024-06-16 ... | CF005 | 1 | 5 | 5 | 125.0 | 2.0 |
| 6 | 5 | 5 | INV005 | 2024-06-16 ... | CF005 | 1 | 6 | 6 | 150.0 | 2.0 |
| 7 | 5 | 5 | INV005 | 2024-06-16 ... | CF005 | 1 | 7 | 7 | 175.0 | 1.0 |
| 8 | 5 | 5 | INV005 | 2024-06-16 ... | CF005 | 1 | 8 | 8 | 200.0 | 1.0 |
| 9 | 5 | 5 | INV005 | 2024-06-16 ... | CF005 | 1 | 9 | 9 | 225.0 | 1.0 |

In these tables, the ID column references the ID column in the USER_ table. Thus, each CUSTOMER and ADMIN must be a USER_.

This means that CUSTOMER and ADMIN tables extend the USER_ table, inheriting its ID attribute. This setup ensures that a CUSTOMER or an ADMIN entry cannot exist without a corresponding USER_ entry.

## Transactions

In various parts of the project, transactions have been utilized. About five transactions have been used, and some of these transactions are also used in the trigger section. Below is an example trigger that creates a database:

```
START TRANSACTION;


INSERT INTO USER_ (ID, NAME, EMAIL) VALUES
(1, 'Ali', 'ali@example.com'),
(2, 'Ayşe', 'ayse@example.com'),
(3, 'Mehmet', 'mehmet@example.com'),
(4, 'Fatma', 'fatma@example.com'),
(5, 'Ahmet', 'ahmet@example.com'),
(6, 'Zeynep', 'zeynep@example.com'),
(7, 'Mustafa', 'mustafa@example.com'),
(8, 'Emine', 'emine@example.com'),
(9, 'Burak', 'burak@example.com'),
(10, 'Seda', 'seda@example.com');

INSERT INTO CUSTOMER (ID, LOYALTYPOINTS) VALUES
(1, 100),
(2, 200),
(3, 300),
(4, 400),
(5, 500);

INSERT INTO ADMIN (ID, ROLE) VALUES
(6, 'Manager'),
(7, 'Supervisor'),
(8, 'Admin');
```

```sql
INSERT INTO ADDRESS (ID, COUNTRYID, CITYID, TOWNID, DISTRICTID, POSTALCODE, ADDRESSTEXT, USERID) VALUES

(1, 1, 1, 1, 1, '34710', 'Example Address 1', 1),

(2, 1, 1, 1, 2, '34720', 'Example Address 2', 2),

(3, 1, 1, 2, 1, '34730', 'Example Address 3', 3),

(4, 2, 2, 2, 2, '11211', 'Example Address 4', 4),

(5, 2, 2, 2, 1, '11222', 'Example Address 5', 5);


INSERT INTO COUNTRY (ID, COUNTRY) VALUES

(1, 'Turkey'),

(2, 'USA');


INSERT INTO CITY (ID, CITY) VALUES

(1, 'Istanbul'),

(2, 'New York');


INSERT INTO TOWN (ID, TOWN) VALUES

(1, 'Kadikoy'),

(2, 'Brooklyn');


INSERT INTO DISTRICT (ID, DISTRICT) VALUES

(1, 'Moda'),

(2, 'Williamsburg');


INSERT INTO ITEM (ID, ITEMCODE, ITEMNAME, PRICE, CATEGORY1, CATEGORY2, CATEGORY3) VALUES

(1, 'ITEM001', 'Sample Item 1', 25.0, 'Category1', 'Category2', 'Category3'),

(2, 'ITEM002', 'Sample Item 2', 50.0, 'Category1', 'Category2', 'Category3'),

(3, 'ITEM003', 'Sample Item 3', 75.0, 'Category1', 'Category2', 'Category3'),

(4, 'ITEM004', 'Sample Item 4', 100.0, 'Category1', 'Category2', 'Category3'),

(5, 'ITEM005', 'Sample Item 5', 125.0, 'Category1', 'Category2', 'Category3'),

(6, 'ITEM006', 'Sample Item 6', 150.0, 'Category1', 'Category2', 'Category3'),

(7, 'ITEM007', 'Sample Item 7', 175.0, 'Category1', 'Category2', 'Category3'),

(8, 'ITEM008', 'Sample Item 8', 200.0, 'Category1', 'Category2', 'Category3'),

(9, 'ITEM009', 'Sample Item 9', 225.0, 'Category1', 'Category2', 'Category3'),

(10, 'ITEM010', 'Sample Item 10', 250.0, 'Category1', 'Category2', 'Category3');


INSERT INTO BASKET (ID, USERID, CREATEDDATE, LASTMODIFIEDDATE, ITEMCOUNT, TOTALPRICE, STATUS_) VALUES

(1, 1, NOW(), NOW(), 2, 50.0, 1),
(2, 2, NOW(), NOW(), 1, 50.0, 1),

(3, 3, NOW(), NOW(), 3, 150.0, 1),

(4, 4, NOW(), NOW(), 4, 300.0, 1),

(5, 5, NOW(), NOW(), 5, 500.0, 1);


INSERT INTO BASKETDETAIL (ID, BASKETID, ITEMID, AMOUNT, PRICE, TOTALPRICE, DATE_)
VALUES

(1, 1, 1, 2, 25.0, 50.0, NOW()),

(2, 2, 2, 1, 50.0, 50.0, NOW()),

(3, 3, 3, 1, 75.0, 75.0, NOW()),

(4, 3, 4, 2, 50.0, 100.0, NOW()),

(5, 4, 5, 2, 125.0, 250.0, NOW()),

(6, 4, 6, 2, 150.0, 300.0, NOW()),

(7, 5, 7, 1, 175.0, 175.0, NOW()),

(8, 5, 8, 1, 200.0, 200.0, NOW()),

(9, 5, 9, 1, 225.0, 225.0, NOW());


INSERT INTO PAYMENT (ID, BASKETID, TOTALPRICE, PAYMENTTYPE, DATE_, ISOK, APPROVECODE,
ERROR_) VALUES

(1, 1, 50.0, 1, NOW(), 1, 'APPROVED', ''),

(2, 2, 50.0, 1, NOW(), 1, 'APPROVED', ''),

(3, 3, 150.0, 2, NOW(), 1, 'APPROVED', ''),

(4, 4, 300.0, 2, NOW(), 1, 'APPROVED', ''),

(5, 5, 500.0, 1, NOW(), 1, 'APPROVED', '');


INSERT INTO ORDER (ID, USERID, BASKETID, CREATEDDATE, ITEMCOUNT, TOTALPRICE, STATUS_)
VALUES

(1, 1, 1, NOW(), 2, 50.0, 1),

(2, 2, 2, NOW(), 1, 50.0, 1),

(3, 3, 3, NOW(), 3, 150.0, 1),

(4, 4, 4, NOW(), 4, 300.0, 1),

(5, 5, 5, NOW(), 5, 500.0, 1);


INSERT INTO ORDERDETAIL (ID, ORDERID, BASKETDETAILID, ITEMID, AMOUNT, PRICE,
TOTALPRICE) VALUES

(1, 1, 1, 1, 2, 25.0, 50.0),

(2, 2, 2, 2, 1, 50.0, 50.0),

(3, 3, 3, 3, 1, 75.0, 75.0),

(4, 3, 4, 4, 2, 50.0, 100.0),

(5, 4, 5, 5, 2, 125.0, 250.0),
(6, 4, 6, 6, 2, 150.0, 300.0),
(7, 5, 7, 7, 1, 175.0, 175.0),
(8, 5, 8, 8, 1, 200.0, 200.0),
(9, 5, 9, 9, 1, 225.0, 225.0);

INSERT INTO INVOICE (ID, ORDERID, INVOICENO, DATE_, CARGOFICHENO, STATUS_) VALUES
(1, 1, 'INV001', NOW(), 'CF001', 1),
(2, 2, 'INV002', NOW(), 'CF002', 1),
(3, 3, 'INV003', NOW(), 'CF003', 1),
(4, 4, 'INV004', NOW(), 'CF004', 1),
(5, 5, 'INV005', NOW(), 'CF005', 1);

INSERT INTO INVOICEDETAIL (ID, INVOICEID, ORDERDETAILID, ITEMID, PRICE, AMOUNT) VALUES
(1, 1, 1, 1, 25.0, 2),
(2, 2, 2, 2, 50.0, 1),
(3, 3, 3, 3, 75.0, 1),
(4, 4, 4, 4, 50.0, 2),
(5, 5, 5, 5, 125.0, 2),
(6, 5, 6, 6, 150.0, 2),
(7, 5, 7, 7, 175.0, 1),
(8, 5, 8, 8, 200.0, 1),
(9, 5, 9, 9, 225.0, 1);

COMMIT;
```

Another transaction:

```
START TRANSACTION;
INSERT INTO USER_ (NAME, EMAIL) VALUES ('Ali Veli', 'ali.veli@example.com');
SET @user_id = LAST_INSERT_ID();
INSERT INTO CUSTOMER (ID, LOYALTYPOINTS) VALUES (@user_id, 0);
INSERT INTO BASKET (USERID, CREATEDDATE, LASTMODIFIEDDATE, ITEMCOUNT, TOTALPRICE, STATUS_)
VALUES (@user_id, NOW(), NOW(), 0, 0.0, 1);
SET @basket_id = LAST_INSERT_ID();
INSERT INTO ITEM (ITEMCODE, ITEMNAME, PRICE, CATEGORY1, CATEGORY2, CATEGORY3)
VALUES ('P1234', 'Kalem', 5.0, 'Kırtasiye', 'Yazı Gereçleri', 'Kalemler');
SET @item_id = LAST_INSERT_ID();
INSERT INTO BASKETDETAIL (BASKETID, ITEMID, AMOUNT, PRICE, TOTALPRICE, DATE_)
VALUES (@basket_id, @item_id, 2, 5.0, 10.0, NOW());
SET @basket_detail_id = LAST_INSERT_ID();
```

```
UPDATE BASKET SET ITEMCOUNT = 1, TOTALPRICE = 10.0, LASTMODIFIEDDATE = NOW() WHERE ID =
@basket_id;

INSERT INTO `ORDERS` (USERID, BASKETID, CREATEDDATE, ITEMCOUNT, TOTALPRICE, STATUS_)
VALUES (@user_id, @basket_id, NOW(), 1, 10.0, 1);
SET @orders_id = LAST_INSERT_ID();

INSERT INTO ORDERDETAIL (ORDERID, BASKETDETAILID, ITEMID, AMOUNT, PRICE, TOTALPRICE)
VALUES (@order_id, @basket_detail_id, @item_id, 2, 5.0, 10.0);

INSERT INTO INVOICE (ORDERID, INVOICENO, DATE_, CARGOFICHENO, STATUS_)
VALUES (@order_id, 'INV20230001', NOW(), 'CF20230001', 1);
SET @invoice_id = LAST_INSERT_ID();

INSERT INTO INVOICEDETAIL (INVOICEID, ORDERDETAILID, ITEMID, PRICE, AMOUNT)
VALUES (@invoice_id, @order_id, @item_id, 5.0, 2);

INSERT INTO PAYMENT (BASKETID, TOTALPRICE, PAYMENTTYPE, DATE_, ISOK, APPROVECODE, ERROR_)
VALUES (@basket_id, 10.0, 1, NOW(), 1, 'APPROVED123', NULL);

COMMIT;
```

## Another transactions:

```
START TRANSACTION;
INSERT INTO BASKET (USERID, CREATEDDATE, LASTMODIFIEDDATE, ITEMCOUNT, TOTALPRICE, STATUS_)
VALUES (1, NOW(), NOW(), 0, 0.0, 1);
SET @basket_id = LAST_INSERT_ID();
INSERT INTO ITEM (ITEMCODE, ITEMNAME, PRICE, CATEGORY1, CATEGORY2, CATEGORY3)
VALUES ('P1234', 'Kalem', 5.0, 'Kırtasiye', 'Yazı Gereçleri', 'Kalemler');
SET @item_id = LAST_INSERT_ID();
INSERT INTO BASKETDETAIL (BASKETID, ITEMID, AMOUNT, PRICE, TOTALPRICE, DATE_)
VALUES (@basket_id, @item_id, 2, 5.0, 10.0, NOW());
SET @basket_detail_id = LAST_INSERT_ID();
UPDATE BASKET
SET ITEMCOUNT = 1, TOTALPRICE = 10.0, LASTMODIFIEDDATE = NOW()
WHERE ID = @basket_id;
INSERT INTO `ORDERS` (USERID, BASKETID, CREATEDDATE, ITEMCOUNT, TOTALPRICE, STATUS_)
VALUES (1, @basket_id, NOW(), 1, 10.0, 1);
COMMIT;
```

# Concurrency Control

## 1. Locking Mechanisms

Locking is used to prevent data from being modified by other transactions during a transaction. There are two types of locks: shared and exclusive. A shared lock allows multiple transactions to read the data simultaneously, while an exclusive lock permits only one transaction to modify the data.

**Example:**

When a user adds items to their basket, an exclusive lock on the basket ensures no other transaction can modify it until the current transaction is complete.

```
BEGIN;
SELECT * FROM BASKET WHERE ID = 1 FOR UPDATE;

INSERT INTO BASKETDETAIL (BASKETID, ITEMID, AMOUNT, PRICE, TOTALPRICE, DATE_)
VALUES (1, 101, 2, 50.0, 100.0, NOW());
COMMIT;
```

## 2. Timestamp Ordering

Timestamps ensure transactions are processed in the order they are initiated, preventing conflicts. Each transaction gets a timestamp and is executed based on this order.

**Example:**

When two users try to purchase the same item simultaneously, transactions are ordered by their timestamps, giving priority to the earlier transaction.

```
BEGIN;
SELECT * FROM ORDERDETAIL WHERE ID = 101 AND AMOUNT > 0;

UPDATE ITEM SET AMOUNT = AMOUNT - 1 WHERE ID = 101;
COMMIT;

BEGIN;
SELECT * FROM ITEM WHERE ID = 101 AND AMOUNT > 0;
ROLLBACK;
```

## 3. Multiversion Concurrency Control (MVCC)

MVCC keeps multiple versions of data, allowing read transactions to occur without being blocked by write transactions. This technique is particularly effective for high concurrency read operations.

**Example:**

While one user views the product list, another user can update the product stock simultaneously. Users see the version of data before the update.

```
SELECT * FROM ITEM WHERE CATEGORY1 = 'Cotegory1';

BEGIN;
UPDATE ITEM SET STOCK = STOCK + 10 WHERE ID = 101;
COMMIT;
```

These techniques provide effective ways to handle concurrency issues in database systems. Employing the correct method enhances database performance and maintains data integrity.

## Inheritance

Inheritance is when a table inherits the properties of another table. In a database schema, this is achieved by linking a table to another table using a foreign key. In this scenario, the CUSTOMER and ADMIN tables inherit from the USER_ table. These tables connect to the USER_ table using the ID column and utilize the information in the USER_ table.

CUSTOMER and ADMIN tables:

```
CREATE TABLE CUSTOMER (
    ID INT,
    LOYALTYPOINTS INT,
    PRIMARY KEY (ID),
    FOREIGN KEY (ID) REFERENCES USER_(ID)
);

CREATE TABLE ADMIN (
    ID INT,
    ROLE VARCHAR(50),
    PRIMARY KEY (ID),
    FOREIGN KEY (ID) REFERENCES USER_(ID)
);
```

## Privileges and Roles

Privileges and roles in a database are essential for defining and controlling user access and actions on the data. Proper management of these elements is crucial for ensuring database security and data integrity.

Roles are used to group together a set of privileges that a user or group of users need to perform specific tasks or functions. For example, an "Admin" role might have full access to all database functionalities, whereas a "Customer" role might only have permissions to view and update their personal information. This grouping simplifies the assignment of permissions and ensures that users have the appropriate access for their roles within the organization.

Privileges define the specific actions that can be performed on database objects, such as tables or views. These actions can include operations like "SELECT", "INSERT", "UPDATE", and "DELETE". By assigning privileges carefully, unauthorized access and potential data breaches can be prevented. For instance, a user might be granted "SELECT" and "INSERT" privileges on a table, allowing them to view and add data, but not modify or delete existing records.

In summary, roles and privileges are vital for ensuring that users have the correct access levels needed to perform their duties. This not only protects sensitive data from unauthorized access but also helps maintain the overall security and efficiency of database operations.

## Additional Details:

This database structure is designed for an e-commerce platform. Users can be either customers or administrators, and each user can have multiple addresses. Countries, cities, towns, and districts are organized hierarchically. Products are categorized into various categories, and each basket and order is tracked in detail. Payments and invoices are also recorded and managed within this system.

**Business Rules and Constraints:**

- Each user can have one or more addresses.
- Each customer and administrator must be a user.
- A basket can contain multiple products and must be created by a user.
- Orders are created based on the items in the basket and are linked to a user and a basket.
- Payments are made based on the total price in the basket and are linked to a basket.
- Invoices are generated from orders and are linked to an order.
- Each product can belong to multiple categories.
- Payments have fields for an approval code and error message to indicate success or failure.

## SQL Statements:

### Database Creation:

```
CREATE TABLE USER_ (
    ID INT PRIMARY KEY,
    NAME VARCHAR(50),
    EMAIL VARCHAR(100)
);

CREATE TABLE CUSTOMER (
    ID INT PRIMARY KEY,
    LOYALTYPOINTS INT,
    FOREIGN KEY (ID) REFERENCES USER_(ID)
);

CREATE TABLE ADMIN (
    ID INT PRIMARY KEY,
    ROLE VARCHAR(50),
    FOREIGN KEY (ID) REFERENCES USER_(ID)
);

CREATE TABLE ADDRESS (
    ID INT PRIMARY KEY,
    COUNTRYID INT,
    CITYID INT,
    TOWNID INT,
    DISTRICTID INT,
    POSTALCODE VARCHAR(10),
    ADDRESSTEXT VARCHAR(250),
    USERID INT,
    FOREIGN KEY (USERID) REFERENCES USER_(ID)
);

CREATE TABLE COUNTRY (
    ID INT PRIMARY KEY,
    COUNTRY VARCHAR(100)
);

CREATE TABLE CITY (
    ID INT PRIMARY KEY,
    CITY VARCHAR(100)
);

CREATE TABLE TOWN (
    ID INT PRIMARY KEY,
    TOWN VARCHAR(100)
);

CREATE TABLE DISTRICT (
    ID INT PRIMARY KEY,
    DISTRICT VARCHAR(100)
);

CREATE TABLE ITEM (
    ID INT PRIMARY KEY,
    ITEMCODE VARCHAR(20),
    ITEMNAME VARCHAR(100),
    PRICE FLOAT,
    CATEGORY1 VARCHAR(50),
    CATEGORY2 VARCHAR(50),
    CATEGORY3 VARCHAR(50)
);

CREATE TABLE BASKET (
    ID INT PRIMARY KEY,
    USERID INT,
    CREATEDDATE DATETIME,
    LASTMODIFIEDDATE DATETIME,
    ITEMCOUNT INT,
    TOTALPRICE FLOAT,
    STATUS_ INT,
    FOREIGN KEY (USERID) REFERENCES USER_(ID)
);

CREATE TABLE BASKETDETAIL (
    ID INT PRIMARY KEY,
    BASKETID INT,
    ITEMID INT,
    AMOUNT FLOAT,
    PRICE FLOAT,
    TOTALPRICE FLOAT,
    DATE_ DATETIME,
    FOREIGN KEY (BASKETID) REFERENCES BASKET(ID),
    FOREIGN KEY (ITEMID) REFERENCES ITEM(ID)
);



CREATE TABLE PAYMENT (
    ID INT PRIMARY KEY,
    BASKETID INT,
    TOTALPRICE FLOAT,
    PAYMENTTYPE INT,
    DATE_ DATETIME,
    ISOK BIT,
    APPROVECODE VARCHAR(20),
    ERROR_ VARCHAR(1000),
    FOREIGN KEY (BASKETID) REFERENCES BASKET(ID)
);
```

```
CREATE TABLE ORDER (
    ID INT PRIMARY KEY,
    USERID INT,
    BASKETID INT,
    CREATEDDATE DATETIME,
    ITEMCOUNT INT,
    TOTALPRICE FLOAT,
    STATUS_ INT,
    FOREIGN KEY (USERID) REFERENCES USER_(ID),
    FOREIGN KEY (BASKETID) REFERENCES BASKET(ID)
);

CREATE TABLE ORDERDETAIL (
    ID INT PRIMARY KEY,
    ORDERID INT,
    BASKETDETAILID INT,
    ITEMID INT,
    AMOUNT FLOAT,
    PRICE FLOAT,
    TOTALPRICE FLOAT,
    FOREIGN KEY (ORDERID) REFERENCES ORDER(ID),
    FOREIGN KEY (BASKETDETAILID) REFERENCES BASKETDETAIL(ID),
    FOREIGN KEY (ITEMID) REFERENCES ITEM(ID)
);

CREATE TABLE INVOICE (
    ID INT PRIMARY KEY,
    ORDERID INT,
    INVOICENO VARCHAR(50),
    DATE_ DATETIME,
    CARGOFICHENO VARCHAR(50),
    STATUS_ INT,
    FOREIGN KEY (ORDERID) REFERENCES ORDER(ID)
);




CREATE TABLE INVOICEDETAIL (
    ID INT PRIMARY KEY,
    INVOICEID INT,
    ORDERDETAILID INT,
    ITEMID INT,
    PRICE FLOAT,
    AMOUNT FLOAT,
    FOREIGN KEY (INVOICEID) REFERENCES INVOICE(ID),
    FOREIGN KEY (ORDERDETAILID) REFERENCES ORDERDETAIL(ID),
    FOREIGN KEY (ITEMID) REFERENCES ITEM(ID)
);
```

**Data Manipulation and Management:**

1. **Add New User:**

   ```
   INSERT INTO USER_ (ID, NAME, EMAIL) VALUES (1, 'John
   Doe', 'john@example.com');
   ```

2. **Add New Product:**

   ```
   INSERT INTO ITEM (ID, ITEMCODE, ITEMNAME, PRICE,
   CATEGORY1) VALUES (1, 'A001', 'Laptop', 1500.00,
   'Electronics');
   ```

3. **Add Item to Basket:**

   ```
   INSERT INTO BASKETDETAIL (ID, BASKETID, ITEMID, AMOUNT,
   PRICE, TOTALPRICE, DATE_)
   VALUES (1, 1, 1, 1, 1500.00, 1500.00, GETDATE());
   ```

4. **Create Order:**

   ```
   INSERT INTO ORDER (ID, USERID, BASKETID, CREATEDDATE,
   ITEMCOUNT, TOTALPRICE, STATUS_)
   VALUES (1, 1, 1, GETDATE(), 1, 1500.00, 1);
   ```

5. **Process Payment:**

   ```
   INSERT INTO PAYMENT (ID, BASKETID, TOTALPRICE,
   PAYMENTTYPE, DATE_, ISOK, APPROVECODE)
   VALUES (1, 1, 1500.00, 1, GETDATE(), 1, 'APPROVED');
   ```

6. **Generate Invoice:**

   ```
   INSERT INTO INVOICE (ID, ORDERID, INVOICENO, DATE_,
   CARGOFICHENO, STATUS_)
   VALUES (1, 1, 'INV001', GETDATE(), 'CF001', 1);
   ```

These SQL statements and the database structure cover all the essential operations and management required for an e-commerce platform.