GEBZE TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Databases Course (CSE 414)
Project

PREPARER

171044073 Berkan AKIN

ADVISOR

Dr. Öğr. Üyesi Burcu YILMAZ

Spring Semester, 2023

GEBZE/KOCAELİ

Contents

Introduction

This report is prepared to evaluate a developed e-commerce web application. The web application is a user-friendly e-commerce platform that enables customers to easily shop online. The purpose of this report is to provide a detailed examination of the database tables created for the project and an overview of the general functioning of the web application.

Database Tables: The designed database for the project includes a set of tables that store users, addresses, products, baskets, orders, invoices, and other important information. Here is an overview of these tables:

USER_ Table: This table contains user information such as username, password, name, email, gender, birthdate, and phone numbers.

ADDRESS Table: This table holds user address information. It includes country, city, district, postal code, and street address. It is related to the USER_ table.

COUNTRY Table: This table stores country names and is related to the ADDRESS table. It allows users to select their country during the address entry.

ITEM Table: The ITEM table contains information about the products available on the website. It includes item code, name, price, and category.

BASKET Table: This table represents the user's shopping basket. It stores details such as creation date, last modified date, item count, total price, and status.

BASKETDETAIL Table: This table contains the detailed information of items within the user's basket. It links the basket and item information.

PAYMENT Table: The PAYMENT table stores information related to payment transactions. It includes details like the associated basket, total price, payment type, date, and payment status.

ORDER Table: The ORDER table holds the details of customer orders. It includes user, basket, creation date, item count, total price, and status.

ORDERDETAIL Table: This table contains the details of items within an order. It links the order and item information.

INVOICE Table: The INVOICE table stores the details of invoices. It includes the associated order, invoice number, date, cargo tracking number, and status.

INVOICEDETAIL Table: This table contains the details of items within an invoice. It links the invoice and item information.

General Features of the Web Application:

The web application provides users with a user-friendly interface that allows them to register, log in, and easily search and purchase products. Here are the general features of the web application:

- Users can register and log in to their personal accounts.
- Products are categorized, allowing users to filter and search for specific items.
- The product detail page provides users with detailed information, including price and stock availability.
- Users can add products to their shopping basket, view the basket, and manage the items within it.
- Multiple payment options are available for users to complete their transactions.
- Users can view their order history and access details of their past orders.
- Invoice information is associated with orders, allowing users to view and download their invoices.

Conclusion

This report provided an overview of the database tables created for the project and a general understanding of the web application's functionality. The designed database tables efficiently store user information, addresses, products, baskets, orders, and invoices. The web application offers a user-friendly interface to enhance the shopping experience for customers.

User Requirements

Registration and Login

- Users should be able to create an account on the website by registering.
- During the registration process, basic information such as username, password, name, surname, and email should be requested.
- Once registered, users should be able to log in to their accounts.
- Login process should require username and password.

Profile Management:

- After logging in, users should have the ability to update their profile information.
- Profile information should include fields like name, surname, email, and phone number.
- Users should be able to change their password from their account.

Product Search and Filtering:

- Users should be able to search for products on the website and apply filters.
- Search functionality should allow searching by product name or category.
- Users should be able to filter products based on criteria like price, brand, color, etc.

Product Details

- Users should be able to access the detailed page of a selected product.
- Product details page should display information such as product images, description, price, and stock availability.
- Users should be able to add the product to their shopping cart from the product details page.

Cart Operations

- Users should be able to view, edit, and remove products added to their shopping cart.
- The shopping cart should display the total price and quantity of items.
- Users should be able to review the products in their cart before proceeding to checkout.

Checkout Process

- Users should be able to make payments for the products in their cart.
- Different payment methods should be available, such as credit card, bank transfer, PayPal, etc.
- During the checkout process, users should provide the necessary payment information and confirm their orders.

Order Tracking

- Users should be able to track the status of their placed orders.
- Users should be provided with tracking numbers for their orders.
- Users should be able to view their order history and past orders through their account.
- Returns and Return Tracking
- Users should be able to initiate returns for faulty or unsatisfactory products.
- Return requests should be made through the user's account and tracked accordingly.
- Users should be informed about the return process.

Normalization

First Normal Form (1NF)

- All tables are grouped under a single theme, and each field in the table contains a single value.
- There are no repeating groups or repeated fields within a group in the tables.

Second Normal Form (2NF)

- Ensured that the conditions of 1NF are met.
- Tables are decomposed to identify all key fields and properly define functional dependencies among other fields in the table.
- Relational dependencies are minimized.

Third Normal Form (3NF)

- Ensured that the conditions of 2NF are met.
- All transitive dependencies are eliminated.
- Tables reflect direct relationships among the fields outside of the key fields.

Boyce-Codd Normal Form (BCNF)

- Ensured that the conditions of 3NF are met.
- If any functional dependency exists where any non-key field depends on another non-key field, that dependency is eliminated.
- This level of normalization is more stringent than 3NF and encompasses more complex relational structures.

Fourth Normal Form (4NF)

- Ensured that the conditions of BCNF are met.
- Any multi-valued dependencies are eliminated.
- Each table contains independently processable fields that are not affected by other fields.

These normalization levels are a set of rules used to optimize data integrity and the relational structure of the database. Each level represents a higher level of database normalization, aiming to minimize data redundancy, dependencies, and meaningless relationships.

Functional Dependencies

The main functional dependencies between tables are listed below

USER_ Table

- ID -> (USERNAME_, PASSWORD_, NAMESURNAME, EMAIL, GENDER, CREATEDDATE, BIRTHDATE, TELNR1, TELNR2)
- USERNAME_ -> (PASSWORD_, NAMESURNAME, EMAIL, GENDER, CREATEDDATE, BIRTHDATE, TELNR1, TELNR2)

ADDRESS Table

- ID -> (COUNTRYID, CITYID, TOWNID, DISTRICTID, POSTALCODE, ADDRESSTEXT, USERID)
- USERID -> (COUNTRYID, CITYID, TOWNID, DISTRICTID, POSTALCODE, ADDRESSTEXT)

ITEM Table

- ID -> (ITEMCODE, ITEMNAME, PRICE, CATEGORY1, CATEGORY2, CATEGORY3)

BASKET Table

- ID -> (USERID, CREATEDDATE, LASTMODIFIEDDATE, ITEMCOUNT, TOTALPRICE, STATUS_)

BASKETDETAIL Table

- ID -> (BASKETID, ITEMID, AMOUNT, PRICE, TOTALPRICE, DATE_)

PAYMENT Table

- ID -> (BASKETID, TOTALPRICE, PAYMENTTYPE, DATE_, ISOK, APPROVECODE, ERROR_)

ORDER Table

- ID -> (USERID, BASKETID, CREATEDDATE, ITEMCOUNT, TOTALPRICE, STATUS_)

ORDERDETAIL Table

- ID -> (ORDERID, BASKETDETAILID, ITEMID, AMOUNT, PRICE, TOTALPRICE)

INVOICE Table

- ID -> (ORDERID, INVOICENO, DATE_, CARGOFICHENO, STATUS_)

INVOICEDETAIL Table

- ID -> (INVOICEID, ORDERDETAILID, ITEMID, PRICE, AMOUNT)

Trigers

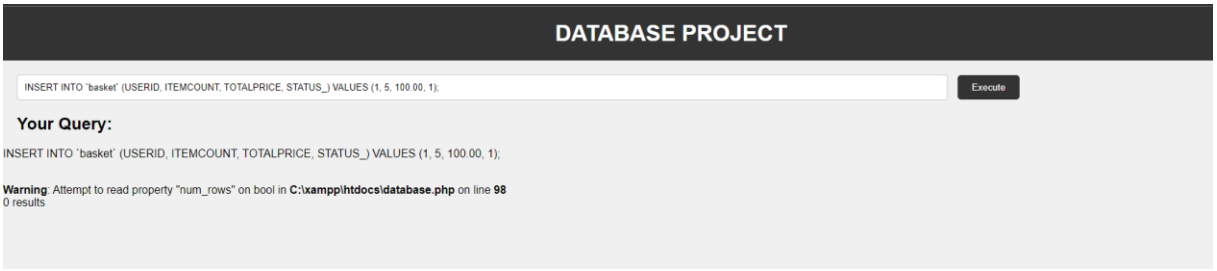
1. Automatically update the creation date of the record when a new record is added in the BASKET table:
   This SQL command adds a new record to the "basket" table. User ID (USERID), number of products (ITEMCOUNT), total price (TOTALPRICE) and status (STATUS_) fields are assigned values. The trigger will automatically update the CREATEDDATE field of this record with the NOW() function.

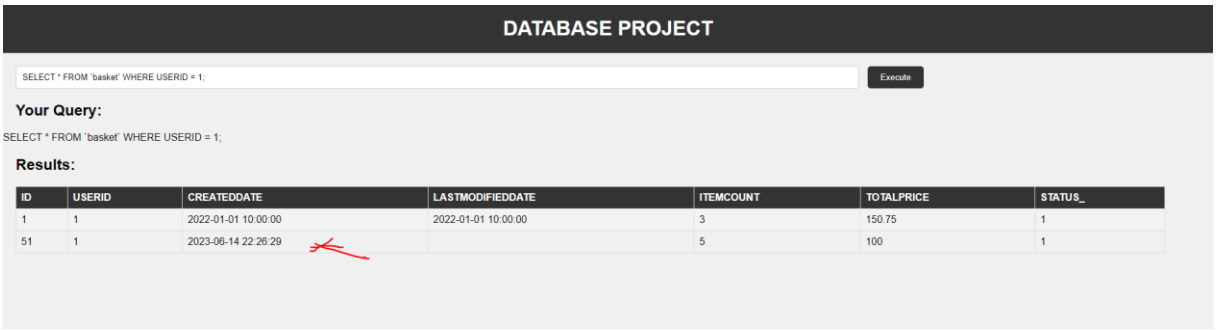
   This trigger automatically updates the CREATEDDATE field when adding a new record to the "basket" table. NEW.CREATEDDATE = NOW(); line assigns the current date and time value to the CREATEDDATE field of the newly added record. So each time you add a new "basket" record, the CREATEDDATE field is automatically updated.
   Ez

```
DELIMITER //

CREATE TRIGGER trg_basket_insert
BEFORE INSERT ON BASKET
FOR EACH ROW
BEGIN
    SET NEW.CREATEDDATE = NOW();
END//

DELIMITER ;
```

Sorguyu güncelle | Sorguyu gönder

Example



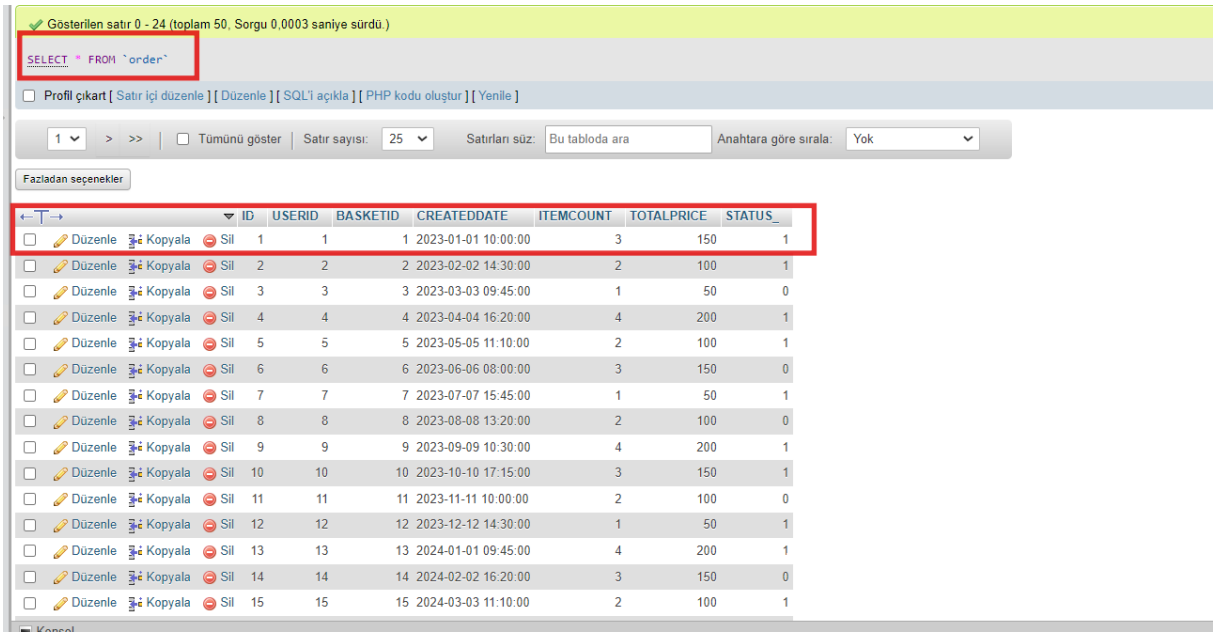


2. Updating the total number of items of the order when a record is deleted in the order detail (ORDERDETAIL)
   This SQL code deletes a record with an ID value of 1 from the "orderdetail" table. This deletion will trigger the trigger.

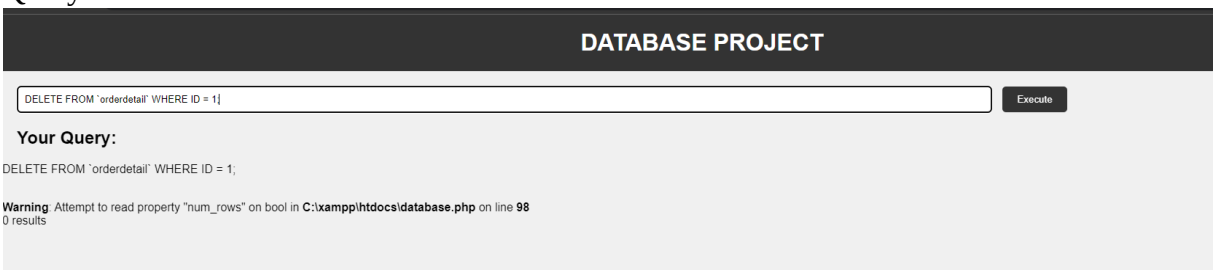
   The action to be taken by the trigger is to update the ITEMCOUNT field in the ORDER table when a record is deleted from the orderdetail table. The trigger finds the corresponding record in the ORDER table using the ORDERID value of the deleted record and updates the ITEMCOUNT field.

```
DELIMITER //

CREATE TRIGGER trg_orderdetail_delete
AFTER DELETE ON ORDERDETAIL
FOR EACH ROW
BEGIN
    UPDATE `ORDER` SET ITEMCOUNT = (SELECT COUNT(*) FROM ORDERDETAIL WHERE ORDERID = OLD.ORDERID) WHERE ID = OLD.ORDERID;
END//

DELIMITER ;
```

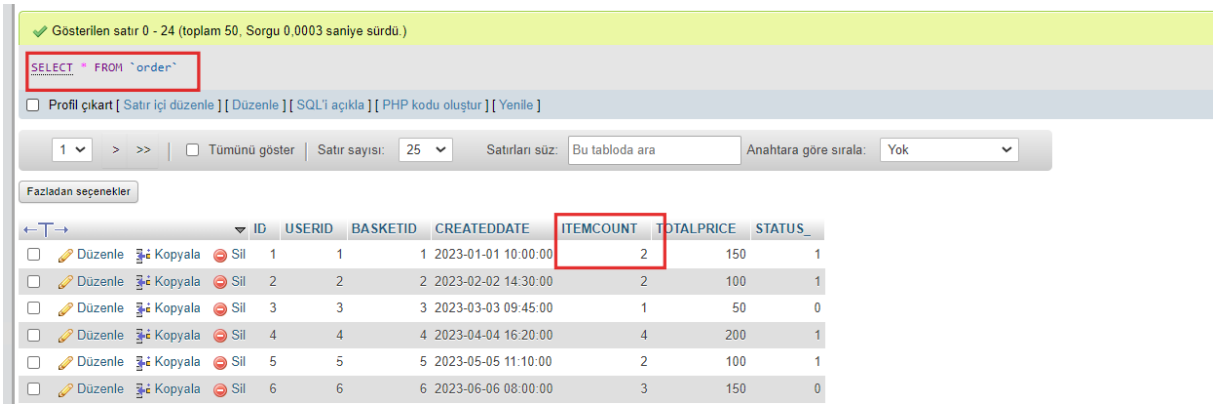
İnitial state Item Count: 3



Query



Latest status Item Count:



4.) Checking the validity of the email address when a record is updated in the User (USER_) table.

This SQL command updates the email (EMAIL) field of the record with an ID value of 1 in the "user_" table to an invalid email address "invalid_email". This update process will trigger the trigger.

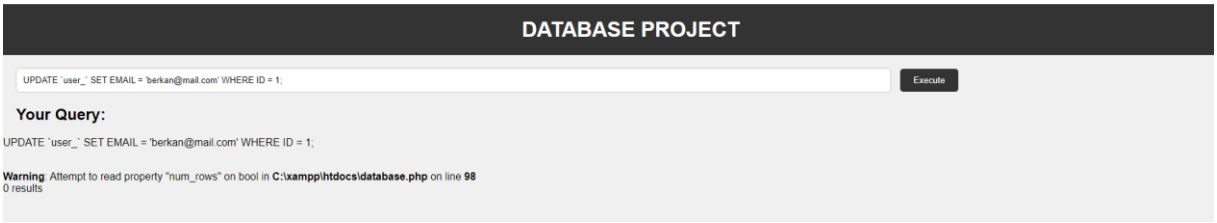
This trigger is when updating a record in the "user_" table (BEFORE UPDATE), a specific pattern (^[A-Za-z0-9._%+-]+@[A-Za) of the new email address (NEW.EMAIL). -z0-9.-]+.[A-Za-z]{2,}$). If the new e-mail address does not provide the layout, it will return a triggering error message, preventing the update process.

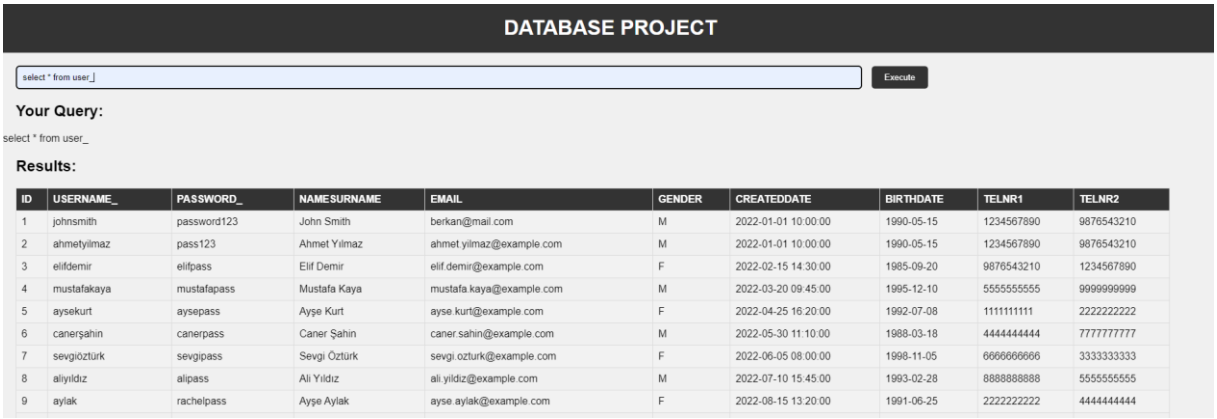
Using the SQL command above you can trigger the trigger and update a record in the "user_" table. This way you can observe what the trigger is doing. If the update process contains an invalid email address, the trigger will return an error message and the update will not occur.

```
DELIMITER //

CREATE TRIGGER trg_user_update
BEFORE UPDATE ON USER_
FOR EACH ROW
BEGIN
    IF NEW.EMAIL NOT REGEXP '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$' THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Geçersiz e-posta adresi';
    END IF;
END//

DELIMITER ;
```

Sorguyu güncelle | Sorguyu gönder

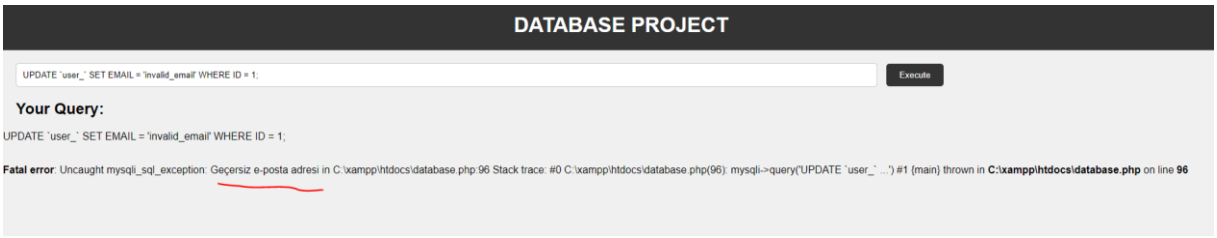
Regular mail update



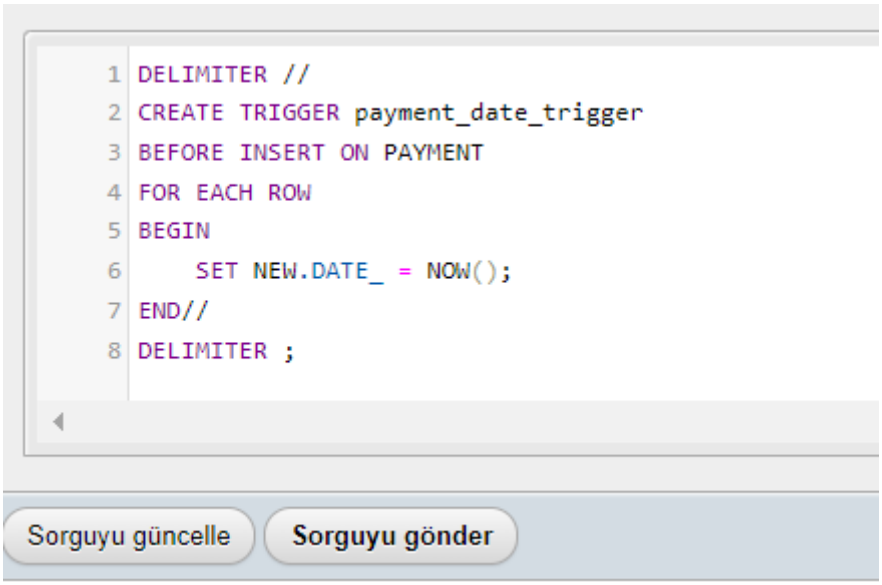


ID	USERNAME_	PASSWORD_	NAMESURNAME	EMAIL	GENDER	CREATEDDATE	BIRTHDATE	TELNR1	TELNR2
1	johnsmith	password123	John Smith	berkan@mail.com	M	2022-01-01 10:00:00	1990-05-15	1234567890	9876543210
2	ahmetyilmaz	pass123	Ahmet Yılmaz	ahmet.yilmaz@example.com	M	2022-01-01 10:00:00	1990-05-15	1234567890	9876543210
3	elifdemir	elifpass	Elif Demir	elif.demir@example.com	F	2022-02-15 14:30:00	1985-09-20	9876543210	1234567890
4	mustafakaya	mustafapass	Mustafa Kaya	mustafa.kaya@example.com	M	2022-03-20 09:45:00	1995-12-10	5555555555	9999999999
5	aysekurt	aysepass	Ayşe Kurt	ayse.kurt@example.com	F	2022-04-25 16:20:00	1992-07-08	1111111111	2222222222
6	canerşahin	canerpass	Caner Şahin	caner.sahin@example.com	M	2022-05-30 11:10:00	1988-03-18	4444444444	7777777777
7	sevgiöztürk	sevgipass	Sevgi Öztürk	sevgi.ozturk@example.com	F	2022-06-05 08:00:00	1998-11-05	6666666666	3333333333
8	aliyıldız	alipass	Ali Yıldız	ali.yildiz@example.com	M	2022-07-10 15:45:00	1993-02-28	8888888888	5555555555
9	aylak	rachelpass	Ayşe Aylak	ayse.aylak@example.com	F	2022-08-15 13:20:00	1991-06-25	2222222222	4444444444

Wrong email update



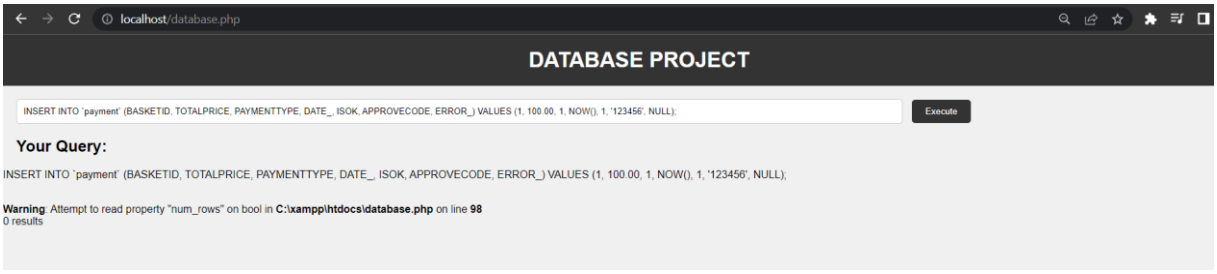
5) Here's a trigger that automatically updates the DATE_ column to the current date and time when adding a new record to the PAYMENT table



Example

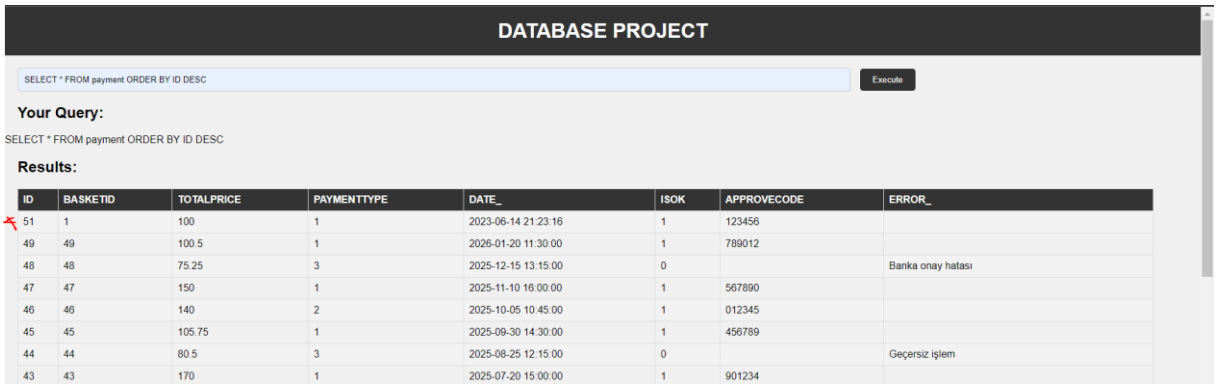
INSERT INTO `payment` (BASKETID, TOTALPRICE, PAYMENTTYPE, DATE_, ISOK, APPROVECODE, ERROR_)

VALUES (1, 100.00, 1, NOW(), 1, '123456', NULL);



Result

Date is the date the trigger was triggered and added. We see it on line 51 with ID.



Join Queries

In SQL, JOIN is an operator used to exchange data between related tables. It allows the merging of two or more tables based on a specific condition and generates a new result set as a result of this merging process.

The JOIN operator performs data merging by utilizing the relationships based on common columns in the tables. Typically, separate tables are created to store data in relational databases, and relationships are established between these tables. For example, there could be a customers table and an orders table.

The JOIN operator enables querying by using these relationships between the tables. By combining two tables, related data can be accessed over a single result set. The JOIN operator can be performed in different types and is commonly used in the following ways:

- **INNER JOIN**: It returns the common records in the tables being joined. In other words, the merging process includes only the rows that have common values.

- **LEFT JOIN**: It returns all the records from the left table and the matching records from the right table. If there are no matching records in the right table, it returns NULL.
- **RIGHT JOIN**: It returns all the records from the right table and the matching records from the left table. If there are no matching records in the left table, it returns NULL.

- **FULL JOIN**: It returns all the records from both the left and right tables. When there are no matching records, it returns NULL.

The JOIN operator is widely used in database queries and allows for effectively merging data to perform more complex and comprehensive queries. By combining data from related tables, we can obtain more comprehensive results.

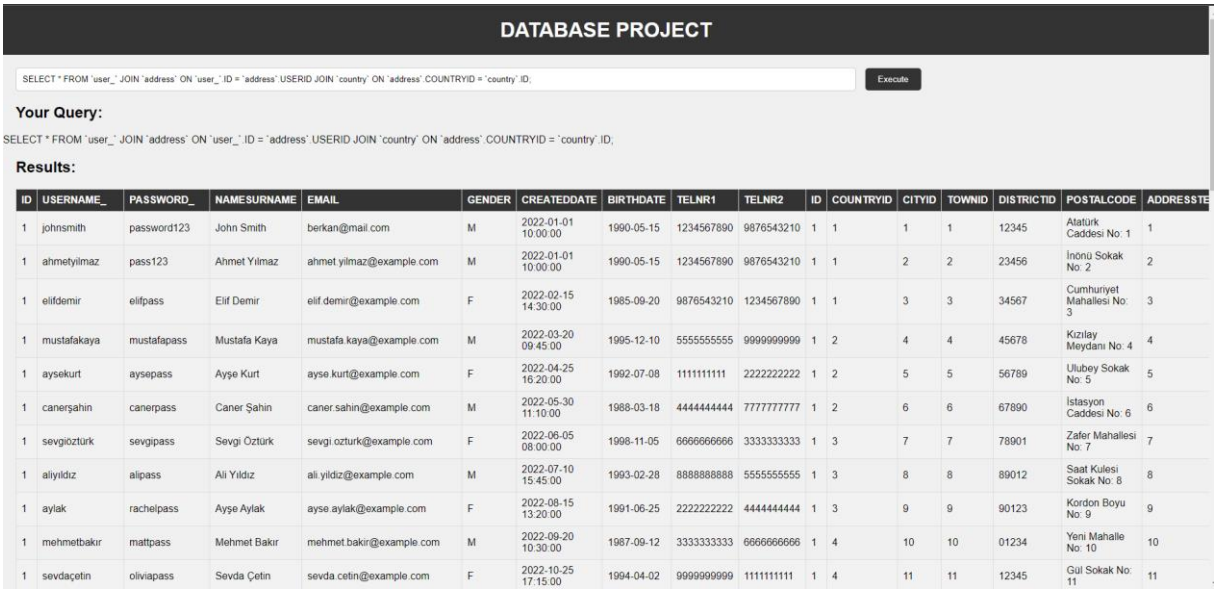
1. INNER JOIN

This query performs an INNER JOIN using the "user_", "address" and "country" tables. The first JOIN operation joins the "user_" and "address" tables via the USERID field. Next, the second JOIN operation joins the "address" and "country" tables via the COUNTRYID field.

This query returns matching records in the "user_", "address", and "country" tables. That is, the addresses of the users and the countries to which these addresses belong are combined. In this way, you can access the country information about the address information of the users.

The INNER JOIN used in this example returns matching records so that only related records are joined.

Example



DATABASE PROJECT

```
SELECT * FROM `user_` JOIN `address` ON `user_`.ID = `address`.USERID JOIN `country` ON `address`.COUNTRYID = `country`.ID;
```

Execute

Your Query:

```
SELECT * FROM `user_` JOIN `address` ON `user_`.ID = `address`.USERID JOIN `country` ON `address`.COUNTRYID = `country`.ID;
```

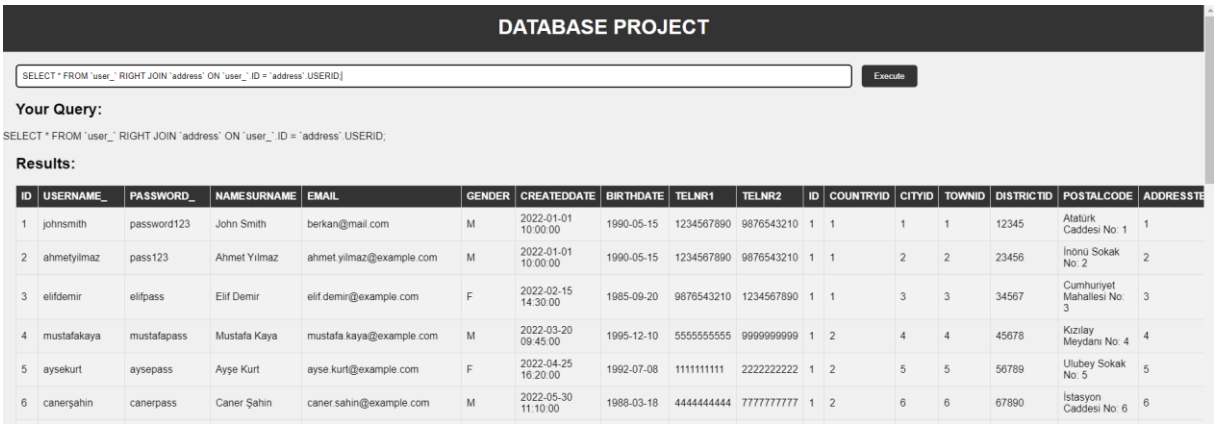
Results:

ID	USERNAME_	PASSWORD_	NAMESURNAME	EMAIL	GENDER	CREATEDDATE	BIRTHDATE	TELNR1	TELNR2	ID	COUNTRYID	CITYID	TOWNID	DISTRICTID	POSTALCODE	ADDRESSTE
1	johnsmith	password123	John Smith	berkan@mail.com	M	2022-01-01 10:00:00	1990-05-15	1234567890	9876543210	1	1	1	1	12345	Atatürk Caddesi No: 1	1
1	ahmetyilmaz	pass123	Ahmet Yılmaz	ahmet.yilmaz@example.com	M	2022-01-01 10:00:00	1990-05-15	1234567890	9876543210	1	1	2	2	23456	İnönü Sokak No: 2	2
1	elifdemir	elifpass	Elif Demir	elif.demir@example.com	F	2022-02-15 14:30:00	1985-09-20	9876543210	1234567890	1	1	3	3	34567	Cumhuriyet Mahallesi No: 3	3
1	mustafakaya	mustafapass	Mustafa Kaya	mustafa.kaya@example.com	M	2022-03-20 09:45:00	1995-12-10	5555555555	9999999999	1	2	4	4	45678	Kızılay Meydanı No: 4	4
1	aysekurt	aysepass	Ayşe Kurt	ayse.kurt@example.com	F	2022-04-25 16:20:00	1992-07-08	1111111111	2222222222	1	2	5	5	56789	Ulubey Sokak No: 5	5
1	canerşahin	canerpass	Caner Şahin	caner.sahin@example.com	M	2022-05-30 11:10:00	1988-03-18	4444444444	7777777777	1	2	6	6	67890	İstasyon Caddesi No: 6	6
1	sevgiöztürk	sevgipass	Sevgi Öztürk	sevgi.ozturk@example.com	F	2022-06-05 08:00:00	1998-11-05	6666666666	3333333333	1	3	7	7	78901	Zafer Mahallesi No: 7	7
1	aliyildiz	alipass	Ali Yıldız	ali.yildiz@example.com	M	2022-07-10 15:45:00	1993-02-28	8888888888	5555555555	1	3	8	8	89012	Saat Kulesi Sokak No: 8	8
1	aylak	rachelpass	Ayşe Aylak	ayse.aylak@example.com	F	2022-08-15 13:20:00	1991-06-25	2222222222	4444444444	1	3	9	9	90123	Kordon Boyu No: 9	9
1	mehmetbakır	mattpass	Mehmet Bakır	mehmet.bakir@example.com	M	2022-09-20 10:30:00	1987-09-12	3333333333	6666666666	1	4	10	10	01234	Yeni Mahalle No: 10	10
1	sevdaçetin	oliviapass	Sevda Çetin	sevda.cetin@example.com	F	2022-10-25 17:15:00	1994-04-02	9999999999	1111111111	1	4	11	11	12345	Gül Sokak No: 11	11

2. RIGHT JOIN

This query performs a right outer join of the "user_" and "address" tables over the USERID field. This join returns all records in table "user_" and matching records in table "address" corresponding to those records. Non-matching records are filled with NULL values.

This query does a right outer join over the USERID field by joining the "user_" and "address" tables. As a result, all records in the "user_" table and matching records in the corresponding "address" table are returned. Non-matching records are represented by NULL values in fields corresponding to the USERID field of records in the "user_" table.



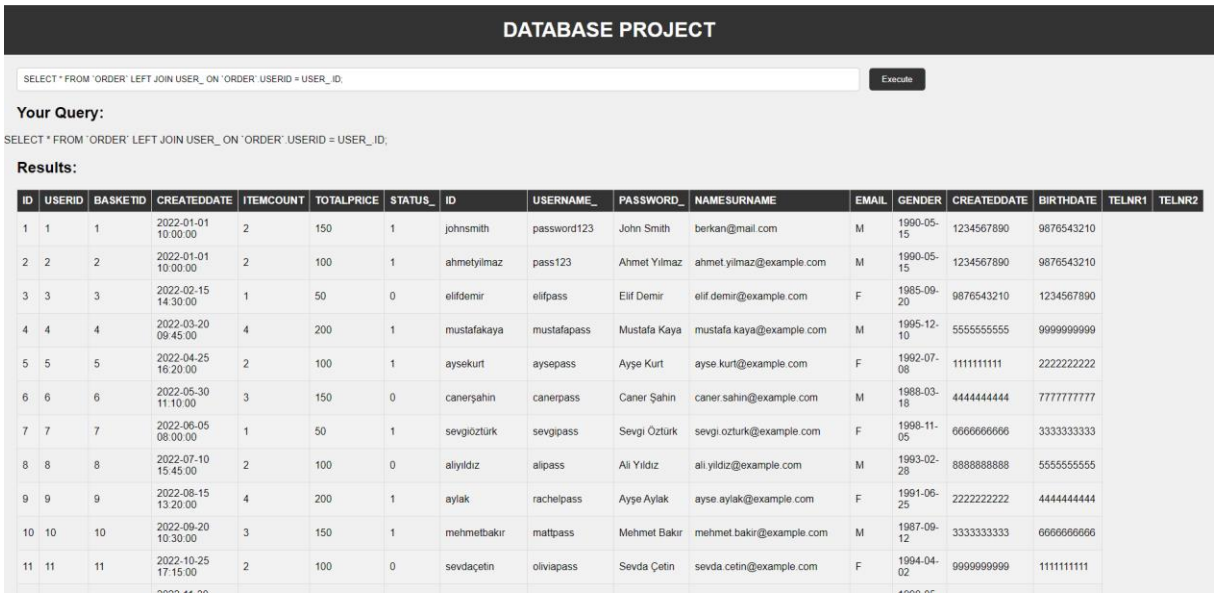
ID	USERNAME_	PASSWORD_	NAMESURNAME	EMAIL	GENDER	CREATEDDATE	BIRTHDATE	TELNR1	TELNR2	ID	COUNTRYID	CITYID	TOWNID	DISTRICTID	POSTALCODE	ADDRESSTE
1	johnsmith	password123	John Smith	berkan@mail.com	M	2022-01-01 10:00:00	1990-05-15	1234567890	9876543210	1	1	1	1	12345	Atatürk Caddesi No: 1	1
2	ahmetyilmaz	pass123	Ahmet Yılmaz	ahmet.yilmaz@example.com	M	2022-01-01 10:00:00	1990-05-15	1234567890	9876543210	1	1	2	2	23456	İnönü Sokak No: 2	2
3	elifdemir	elifpass	Elif Demir	elif.demir@example.com	F	2022-02-15 14:30:00	1985-09-20	9876543210	1234567890	1	1	3	3	34567	Cumhuriyet Mahallesi No: 3	3
4	mustafakaya	mustafapass	Mustafa Kaya	mustafa.kaya@example.com	M	2022-03-20 09:45:00	1995-12-10	5555555555	9999999999	1	2	4	4	45678	Kızılay Meydanı No: 4	4
5	aysekurt	aysepass	Ayşe Kurt	ayse.kurt@example.com	F	2022-04-25 16:20:00	1992-07-08	1111111111	2222222222	1	2	5	5	56789	Ulubey Sokak No: 5	5
6	canerşahin	canerpass	Caner Şahin	caner.sahin@example.com	M	2022-05-30 11:10:00	1988-03-18	4444444444	7777777777	1	2	6	6	67890	İstasyon Caddesi No: 6	6

3. LEFT JOIN

In this example, let's perform a left JOIN using the "ORDER" and "USER_" tables.

This query performs a left JOIN of the "ORDER" and "USER_" tables through the USERID field. As a result, all records in the "ORDER" table and matching records in the corresponding "USER_" table are returned. Non-matching records are represented by NULL values in the fields corresponding to the USERID field of the records in the "USER_" table.

This example joins two tables by performing a left JOIN using the "ORDER" and "USER_" tables. The left JOIN returns all records in the "ORDER" table and the matching records in the corresponding "USER_" table, allowing related information to be joined.

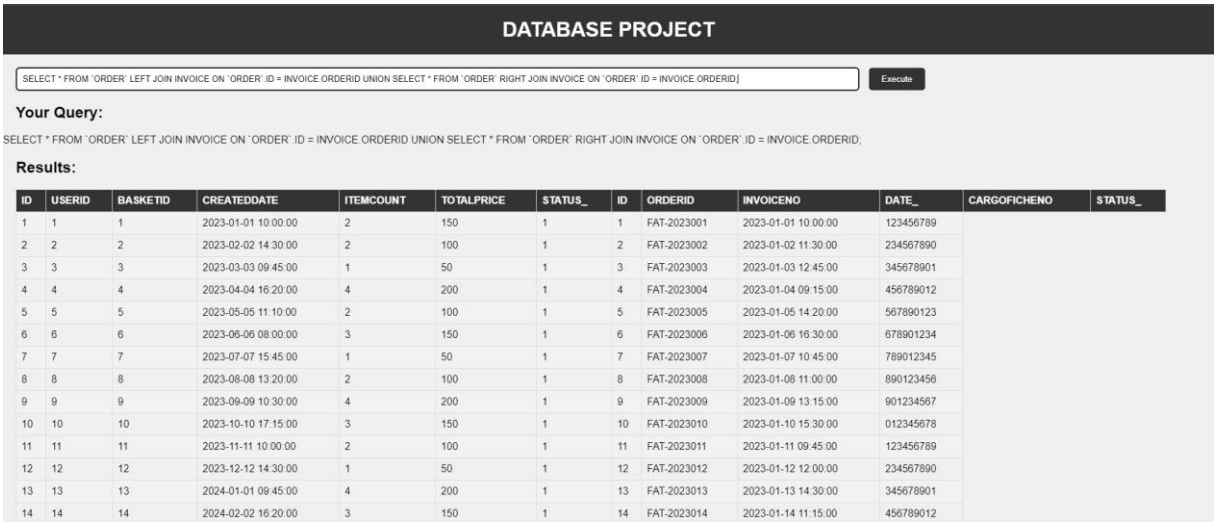


ID	USERID	BASKETID	CREATEDDATE	ITEMCOUNT	TOTALPRICE	STATUS_	ID	USERNAME_	PASSWORD_	NAMESURNAME	EMAIL	GENDER	CREATEDDATE	BIRTHDATE	TELNR1	TELNR2
1	1	1	2022-01-01 10:00:00	2	150	1	johnsmith	password123	John Smith	berkan@mail.com	M	1990-05-15	1234567890	9876543210		
2	2	2	2022-01-01 10:00:00	2	100	1	ahmetyilmaz	pass123	Ahmet Yılmaz	ahmet.yilmaz@example.com	M	1990-05-15	1234567890	9876543210		
3	3	3	2022-02-15 14:30:00	1	50	0	elifdemir	elifpass	Elif Demir	elif.demir@example.com	F	1985-09-20	9876543210	1234567890		
4	4	4	2022-03-20 09:45:00	4	200	1	mustafakaya	mustafapass	Mustafa Kaya	mustafa.kaya@example.com	M	1995-12-10	5555555555	9999999999		
5	5	5	2022-04-25 16:20:00	2	100	1	aysekurt	aysepass	Ayşe Kurt	ayse.kurt@example.com	F	1992-07-08	1111111111	2222222222		
6	6	6	2022-05-30 11:10:00	3	150	0	canerşahin	canerpass	Caner Şahin	caner.sahin@example.com	M	1988-03-18	4444444444	7777777777		
7	7	7	2022-06-05 08:00:00	1	50	1	sevgiöztürk	sevgipass	Sevgi Öztürk	sevgi.ozturk@example.com	F	1998-11-05	6666666666	3333333333		
8	8	8	2022-07-10 15:45:00	2	100	0	aliyıldız	alipass	Ali Yıldız	ali.yildiz@example.com	M	1993-02-28	8888888888	5555555555		
9	9	9	2022-08-15 13:20:00	4	200	1	aylak	rachelpass	Ayşe Aylak	ayse.aylak@example.com	F	1991-06-25	2222222222	4444444444		
10	10	10	2022-09-20 10:30:00	3	150	1	mehmetbakır	mattpass	Mehmet Bakır	mehmet.bakir@example.com	M	1987-09-12	3333333333	6666666666		
11	11	11	2022-10-25 17:15:00	2	100	0	sevdaçetin	oliviapass	Sevda Çetin	sevda.cetin@example.com	F	1994-04-02	9999999999	1111111111		

4. OUTHER JOIN

This example joins the "ORDER" and "INVOICE" tables with a full outer join over the ORDERID field.

This query does a full outer join of the "ORDER" and "INVOICE" tables over the ORDERID field. Left outer join is performed in the first section, while right outer join is performed in the second section. Then the results of the two parts are combined using the UNION operator.



DATABASE PROJECT

```
SELECT * FROM `ORDER` LEFT JOIN INVOICE ON `ORDER`.ID = INVOICE.ORDERID UNION SELECT * FROM `ORDER` RIGHT JOIN INVOICE ON `ORDER`.ID = INVOICE.ORDERID;
```

Execute

Your Query:

SELECT * FROM `ORDER` LEFT JOIN INVOICE ON `ORDER`.ID = INVOICE.ORDERID UNION SELECT * FROM `ORDER` RIGHT JOIN INVOICE ON `ORDER`.ID = INVOICE.ORDERID;

Results:

ID	USERID	BASKETID	CREATEDDATE	ITEMCOUNT	TOTALPRICE	STATUS_	ID	ORDERID	INVOICENO	DATE_	CARGOFICHENO	STATUS_
1	1	1	2023-01-01 10:00:00	2	150	1	1	FAT-2023001	2023-01-01 10:00:00	123456789		
2	2	2	2023-02-02 14:30:00	2	100	1	2	FAT-2023002	2023-01-02 11:30:00	234567890		
3	3	3	2023-03-03 09:45:00	1	50	1	3	FAT-2023003	2023-01-03 12:45:00	345678901		
4	4	4	2023-04-04 16:20:00	4	200	1	4	FAT-2023004	2023-01-04 09:15:00	456789012		
5	5	5	2023-05-05 11:10:00	2	100	1	5	FAT-2023005	2023-01-05 14:20:00	567890123		
6	6	6	2023-06-06 08:00:00	3	150	1	6	FAT-2023006	2023-01-06 16:30:00	678901234		
7	7	7	2023-07-07 15:45:00	1	50	1	7	FAT-2023007	2023-01-07 10:45:00	789012345		
8	8	8	2023-08-08 13:20:00	2	100	1	8	FAT-2023008	2023-01-08 11:00:00	890123456		
9	9	9	2023-09-09 10:30:00	4	200	1	9	FAT-2023009	2023-01-09 13:15:00	901234567		
10	10	10	2023-10-10 17:15:00	3	150	1	10	FAT-2023010	2023-01-10 15:30:00	012345678		
11	11	11	2023-11-11 10:00:00	2	100	1	11	FAT-2023011	2023-01-11 09:45:00	123456789		
12	12	12	2023-12-12 14:30:00	1	50	1	12	FAT-2023012	2023-01-12 12:00:00	234567890		
13	13	13	2024-01-01 09:45:00	4	200	1	13	FAT-2023013	2023-01-13 14:30:00	345678901		
14	14	14	2024-02-02 16:20:00	3	150	1	14	FAT-2023014	2023-01-14 11:15:00	456789012		

Note

- ER diagram is attached.

Used Technologies

- PHP as backend language
- MySQL as database
- HTML CSS Javascript as web interface