
GIT Department of Computer Engineering

CSE414 Databases - Spring 2023

# Project # Report

Okan Torun

1801042662

## 1)Problem Definition

Designing the database management of the e-commerce platform and performing the related operations.

## 2)User Requirements

There are two types of users in the system, Customer and Admin.

**Customers;**

- Can create a new account by entering their email address, first name, last name and address information.
- Can update their profile information.
- Can list products by category or by price range.
- Can create new orders and view order details.
- Can add products to their favorite list.
- Can see reviews and ratings on products.

**Admin;**

- Can list all orders.
- Can add new products to the database with details such as name, description, price, brand, category.
- Can edit existing product information and update product stock levels.
- Can delete products from the database if needed.
- Can add, edit and delete brands and categories.
- Can list a comprehensive view of all orders, including order details, customer information, and order status.

## 3)Database Tables

- Addresses
- Brands
- Categories
- categories_has_products
- Customers
- customers_has_favorite_product
- Items,
- order_items
- Orders,
- Products,
- Reviews
- Price_change_logs
- Premium_customers
- Product_details

## 4)Relations

Customers – addresses (one-to-one)

Customers – orders (one-to-many)

Orders – order_items(one-to-many)

Order_items - items (many-to-one)

Items – products (many-to-one)

Products – brands (many-to-one)

Products – price_change_logs (one-to-many)

Products – categories_has_products (one-to-many)

categories_has_products – categories (many-to-one)

products – reviews (many-to-one)

reviews – customers (many-to-one)

customers_has_favorite_products – customers (many-to-one)

## 5) Normalization

### BCNF:

**Boyce-Codd Normal Form**

- A relation schema $R$ is in BCNF with respect to a set $F$ of functional dependencies if for all functional dependencies in $F^+$ of the form

  $$\alpha \rightarrow \beta$$

  where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - $\alpha$ is a superkey for $R$

**Customer Table:**

Lets apply BCNF rule to **customer** table.

F = id -> email, first_name, last_name, addresses_id

a = id, B = email, first_name, last_name, addresses_id

First rule a -> B is trivial is not satisfied,

we should examine the second rule.

Superkeys of the **customer** table are: id, = email, first_name, last_name, addresses_id

Since id is superkey for R, this table is in Boyce-Codd normal form.

**Addresses Table:**

Lets apply BCNF rule to **customer** table.

F = id -> state, city, street, number

a = id, B = state, city, street, number

First rule a -> B is trivial is not satisfied,

we should examine the second rule.

Superkeys of the **addresses** table are: id, = state, city, street, number

Since id is superkey for R, this table is in Boyce-Codd normal form.

**3NF:**

## Third Normal Form

- A relation schema $R$ is in **third normal form (3NF)** if for all:

  $\alpha \rightarrow \beta$ in $F^+$

  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
  - $\alpha$ is a superkey for $R$
  - Each attribute $A$ in $\beta - \alpha$ is contained in a candidate key for $R$.

    (**NOTE**: each attribute may be in a different candidate key)
- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold).
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).

**Brands:**

Lets apply 3NF rule to brands table.

F = id -> name,description

a = id, B = name,description

First rule a -> B is trivial is not satisfied, we should examine the second rule.

Superkeys of the platforms table are: id,name

Since id is superkey for R, this table is in third normal form. Since one of the rules is satisfied we don't need to examine third rule. It also supports BCNF

**Orders:**

Lets apply 3NF rule to orders table.

F = id -> customer_id,purchase_date

a = id, B = customer_id,purchase_date

First rule a -> B is trivial is not satisfied, we should examine the second rule.

Superkeys of the platforms table are: id,customer_id

Since id is superkey for R, this table is in third normal form. Since one of the rules is satisfied we don't need to examine third rule. It also supports BCNF

## 6) Functional Dependencies

**addresses:**

id -> state, city, street, number

**brands**:

id -> name, description

**categories**:

id -> name, description, parent_category_id

**categories_has_products**:

categories_id -> products_id

**customers**:

id -> email, first_name, last_name, addresses_id

**customers_has_favorite_products**:

customers_id -> products_id

**items**:

id -> production_date, products_id

**order_items**:

orders_id -> items_id

**orders**:

id -> customers_id, purchase_date

**products**:

id -> name, description, price, warranty, brands_id

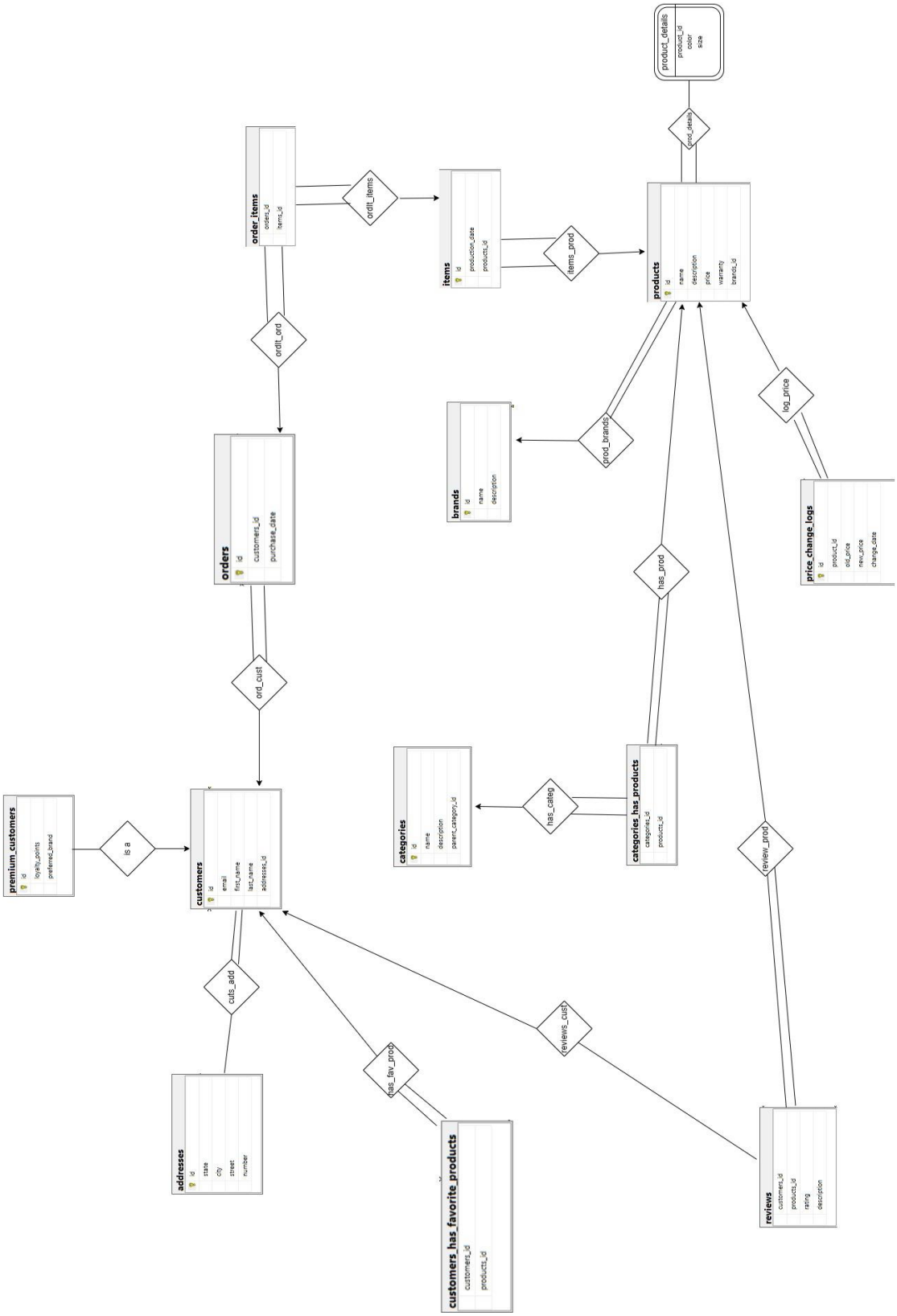
**reviews**:

customers_id, products_id -> rating, description

**premium_customers:**

id,loyaly_points,preffered_brands

# 7)E-R Diagram


product_details
product_id
color
size
prod_details
order_items
orders_id
items_id
ordIt_items
items
id
production_date
products_id
items_prod
products
id
name
description
price
warranty
brands_id
ordIt_ord
prod_brands
log_price
brands
id
name
description
price_change_logs
id
product_id
old_price
new_price
change_date
orders
id
customers_id
purchase_date
has_prod
ord_cust
premium_customers
id
loyalty_points
preferred_brand
is a
customers
id
email
first_name
last_name
addresses_id
categories
id
name
description
parent_category_id
has_categ
categories_has_products
categories_id
products_id
review_prod
custs_add
reviews_cust
has_fav_prod
addresses
id
state
city
street
number
customers_has_favorite_products
customers_id
products_id
reviews
customers_id
products_id
rating
description

## 8-a)Insert-Delete-Update-List Addresses

```
SELECT * FROM addresses
```

| Id | State | City | Street | Number |
|---|---|---|---|---|
| 1 | Turkey | Bursa | papatya | 1 |
| 2 | Turkey | Kocaeli | cengaver | 12 |
| 3 | England | City | Baker | 15 |
| 4 | Turkey | Bursa | Gazi | 3 |
| 13 | Turkey | Düzce | Cenga... | 12 |

```
SELECT * FROM addresses
GO
INSERT INTO addresses (state, city, street, number) VALUES (@State, @City, @Street, @Number)
GO
SELECT * FROM addresses
```

| Id | State | City | Street | Number |
|---|---|---|---|---|
| 1 | Turkey | Bursa | papatya | 1 |
| 2 | Turkey | Kocaeli | cengaver | 12 |
| 3 | England | City | Baker | 15 |
| 4 | Turkey | Bursa | Gazi | 3 |
| 13 | Turkey | Düzce | Cengaver | 12 |
| 14 | Turkey | Balikesir | Cengaver | 12 |

Added a default created value in the interface

```
UPDATE addresses SET city = @NewCity WHERE id = @Id
GO
SELECT * FROM addresses
```

| Id | State | City | Street | Number |
|---|---|---|---|---|
| 1 | Turkey | Bursa | papatya | 1 |
| 2 | Turkey | Kocaeli | cengaver | 12 |
| 3 | England | City | Baker | 15 |
| 4 | Turkey | Bursa | Gazi | 3 |
| 13 | Turkey | Düzce | Cengaver | 12 |
| 14 | Turkey | Çanakkale | Cengaver | 12 |

The update is made according to the entered id and new city information.

The last address city name was changed to Çanakkale.

```
DELETE FROM addresses WHERE id = @Id
GO
SELECT * FROM addresses
```

| Id | State | City | Street | Number |
|---|---|---|---|---|
| 1 | Turkey | Bursa | papatya | 1 |
| 2 | Turkey | Kocaeli | cengaver | 12 |
| 3 | England | City | Baker | 15 |
| 4 | Turkey | Bursa | Gazi | 3 |
| 13 | Turkey | Düzce | Cengaver | 12 |

The address with the entered id is deleted, the address with the number 14 is deleted

## 8-b)Insert-Delete-Update-List Customers

```
SELECT * FROM customers
```

| Id | email | FirstName | LastName | AddressId |
|---|---|---|---|---|
| 1 | asdsad... | okan | torun | 1 |
| 2 | okant... | okann | torunn | 2 |
| 3 | samet | nalbant | s.nalbant@g... | 3 |
| 5 | emir | emir | e.cetin@gma... | 4 |

```
INSERT INTO customers (email, first_name, last_name, addresses_id) VALUES (@Email, @FirstName, @LastName, @AddressId)
GO
SELECT * FROM customers
```

| Id | email | FirstName | LastName | AddressId |
|---|---|---|---|---|
| 1 | asdsad... | okan | torun | 1 |
| 2 | okant... | okann | torunn | 2 |
| 3 | samet | nalbant | s.nalbant@g... | 3 |
| 5 | emir | emir | e.cetin@gma... | 4 |
| 13 | burak... | Burak | Inal | 13 |

```
UPDATE customers SET first_name = @NewFirstName WHERE id = @CustomerId
GO
SELECT * FROM customers
```

| Id | email | FirstName | LastName | AddressId |
|---|---|---|---|---|
| 1 | asdsad... | okan | torun | 1 |
| 2 | okant... | okann | torunn | 2 |
| 3 | samet | nalbant | s.nalbant@g... | 3 |
| 5 | emir | emir | e.cetin@gma... | 4 |
| 13 | burak... | Okan | Inal | 13 |

The last customer's name has been changed.

```
DELETE FROM customers WHERE id = @Id
GO
SELECT * FROM customers
```

| Id | email | FirstName | LastName | AddressId |
|---|---|---|---|---|
| 1 | asdsad... | okan | torun | 1 |
| 2 | okant... | okann | torunn | 2 |
| 3 | samet | nalbant | s.nalbant@g... | 3 |
| 5 | emir | emir | e.cetin@gma... | 4 |

The last customer has been deleted.

## 8-c)Insert-Delete-Update-List Brands

```
SELECT * FROM brands
```

| Id | Name | Descriotion |
|---|---|---|
| 1 | Apple | Apple is very good |
| 2 | Samsung | Smasung is very good |
| 3 | Xiaomi | Xiaomi is very good |
| 4 | Monster | Monster is not good |
| 5 | Toshiba | Toshiba is good |

```
INSERT INTO brands (name, description) VALUES (@Name, @Description)
GO
SELECT * FROM brands
```

| Id | Name | Descriotion |
|---|---|---|
| 1 | Apple | Apple is very good |
| 2 | Samsung | Smasung is very good |
| 3 | Xiaomi | Xiaomi is very good |
| 4 | Monster | Monster is not good |
| 5 | Toshiba | Toshiba is good |
| 8 | Bosch | high quality |

BOSCH is added

```
UPDATE brands SET description = @NewDescription WHERE id = @Id
GO
SELECT * FROM brands
```

| Id | Name | Descriotion |
|---|---|---|
| 1 | Apple | Apple is very good |
| 2 | Samsung | Smasung is very good |
| 3 | Xiaomi | Xiaomi is very good |
| 4 | Monster | Monster is not good |
| 5 | Toshiba | Toshiba is good |
| 9 | Bosch | low quality |

Description updated

```
DELETE FROM brands WHERE id = @Id
GO
SELECT * FROM brands
```

| Id | Name | Descriotion |
|---|---|---|
| 1 | Apple | Apple is very good |
| 2 | Samsung | Smasung is very good |
| 3 | Xiaomi | Xiaomi is very good |
| 4 | Monster | Monster is not good |
| 5 | Toshiba | Toshiba is good |

Last brand is deleted

## 9) Views

### a) Review Views;

Details of review table are shown using View(join). Related Customers and Products tables are connected with Inner Join.

```
SELECT *From reviews
GO
SELECT *From review_details_view
```

Results Messages

| | customers_id | products_id | rating | description |
|---|---|---|---|---|
| 1 | 1 | 7 | 10 | Product is very good |
| 2 | 2 | 8 | 7 | so so |
| 3 | 3 | 9 | 5 | I dont like |
| 4 | 5 | 10 | 8 | it is very good |

| | id | product_name | product_description | price | warranty | first_name | last_name | rating | review_description |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 7 | Iphone 7 | hd camera | 19.99 | 3 | okan | torun | 10 | Product is very good |
| 2 | 8 | Galaxy Note | hd camera | 400.00 | 4 | okann | torunn | 7 | so so |
| 3 | 9 | Xiaomi Mi 6 | 6 GB ram | 300.00 | 2 | nalbant | s.nalbant@gmail.com | 5 | I dont like |
| 4 | 10 | Monster Abra | 1 TB | 500.00 | 4 | emir | e.cetin@gmail.com | 8 | it is very good |

### b)Customer View

The view of the Customer table is provided with Left Outer Join. The Address and Review information of the related Customer is also taken.

```
SELECT *From customers
GO
SELECT * FROM customer_details_view
```

Results Messages

| | id | email | first_name | last_name | addresses_id |
|---|---|---|---|---|---|
| 1 | 1 | asdsad@gmail.com | okan | torun | 1 |
| 2 | 2 | okant@gmail.com | okann | torunn | 2 |
| 3 | 3 | samet | nalbant | s.nalbant@gmail.com | 3 |
| 4 | 5 | emir | emir | e.cetin@gmail.com | 4 |

| | id | email | first_name | last_name | state | city | street | number | rating | description |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | asdsad@gmail.com | okan | torun | Turkey | Bursa | papatya | 1 | 10 | Product is very good |
| 2 | 2 | okant@gmail.com | okann | torunn | Turkey | Kocaeli | cengaver | 12 | 7 | so so |
| 3 | 3 | samet | nalbant | s.nalbant@gmail.com | England | City | Baker | 15 | 5 | I dont like |
| 4 | 5 | emir | emir | e.cetin@gmail.com | Turkey | Bursa | Gazi | 3 | 8 | it is very good |

## c)Product View

The View of the Products table is provided with Left Outer Join. Categories_has_products,categories ,brands ,items information of the related product are also shown together.

```
SELECT *From products
GO
SELECT * FROM product_details_view
```

Results Messages

| | id | name | description | price | warranty | brands_id |
|---|---|---|---|---|---|---|
| 1 | 7 | Iphone 7 | hd camera | 19.99 | 3 | 1 |
| 2 | 8 | Galaxy Note | hd camera | 400.00 | 4 | 2 |
| 3 | 9 | Xiaomi Mi 6 | 6 GB ram | 300.00 | 2 | 3 |
| 4 | 10 | Monster Abra | 1 TB | 500.00 | 4 | 4 |
| 5 | 11 | TSB | it is good | 400.00 | 3 | 5 |

| | id | name | description | price | warranty | category | brand | production_date |
|---|---|---|---|---|---|---|---|---|
| 1 | 7 | Iphone 7 | hd camera | 19.99 | 3 | Electronics | Apple | 2023-06-10 |
| 2 | 8 | Galaxy Note | hd camera | 400.00 | 4 | Electronics | Samsung | 2023-06-10 |
| 3 | 9 | Xiaomi Mi 6 | 6 GB ram | 300.00 | 2 | Electronics | Xiaomi | 2023-06-10 |
| 4 | 10 | Monster Abra | 1 TB | 500.00 | 4 | Electronics | Monster | 2019-06-10 |
| 5 | 11 | TSB | it is good | 400.00 | 3 | Electronics | Toshiba | 2020-06-10 |

## c)Product View With Condition

The same view as above is designed with condition, Production view which is less than production date 2022 is listed.

```
SELECT *From products
GO
SELECT *FROM product_details_with_cond_view
```

Results Messages

| | id | name | description | price | warranty | brands_id |
|---|---|---|---|---|---|---|
| 1 | 7 | Iphone 7 | hd camera | 19.99 | 3 | 1 |
| 2 | 8 | Galaxy Note | hd camera | 400.00 | 4 | 2 |
| 3 | 9 | Xiaomi Mi 6 | 6 GB ram | 300.00 | 2 | 3 |
| 4 | 10 | Monster Abra | 1 TB | 500.00 | 4 | 4 |
| 5 | 11 | TSB | it is good | 400.00 | 3 | 5 |

| | id | name | description | price | warranty | category | brand | production_date |
|---|---|---|---|---|---|---|---|---|
| 1 | 10 | Monster Abra | 1 TB | 500.00 | 4 | Electronics | Monster | 2019-06-10 |
| 2 | 11 | TSB | it is good | 400.00 | 3 | Electronics | Toshiba | 2020-06-10 |

## 10)Triggers

## a) trg_update_product_price

When we update the price of a prodcut here, the trigger kicks in and writes the old and new prices of the product in the price_change_logs table and records the change date.

```
UPDATE products SET price = 19.99 WHERE id = 7
UPDATE products SET price = 29.99 WHERE id = 8
SELECT *FROM price_change_logs
```

| | id | product_id | old_price | new_price | change_date |
|---|---|---|---|---|---|
| 1 | 3 | 7 | 200.00 | 19.99 | 2023-06-12 15:33:04.313 |
| 2 | 4 | 8 | 400.00 | 29.99 | 2023-06-12 15:33:04.313 |

## b)trg_delete_category

Here, when we delete a category in the category table, the product records associated with the deleted category in the categories_has_products table are also deleted.

```
SELECT *FROM categories_has_products
DELETE FROM categories WHERE id = 7
SELECT *FROM categories_has_products
```

Results Messages

| | categories_id | products_id |
|---|---|---|
| 1 | 7 | 7 |
| 2 | 7 | 8 |
| 3 | 7 | 9 |
| 4 | 7 | 10 |
| 5 | 7 | 11 |

| | categories_id | products_id |
|---|---|---|

## c) trg_delete_order

Here, when an order is deleted, the column in the order_items table that the deleted order is associated with is also deleted.

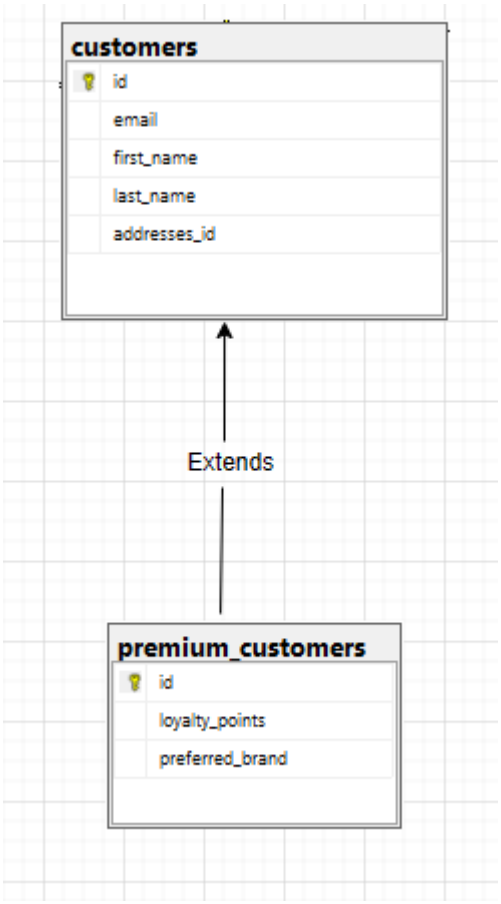
```
SELECT *FROM order_items
DELETE FROM orders WHERE id = 2
SELECT *FROM order_items
```

Results | Messages

| | orders_id | items_id |
|---|---|---|
| 1 | 2 | 2 |
| 2 | 3 | 3 |
| 3 | 4 | 4 |
| 4 | 7 | 5 |

| | orders_id | items_id |
|---|---|---|
| 1 | 3 | 3 |
| 2 | 4 | 4 |
| 3 | 7 | 5 |

## 11) Inheritance

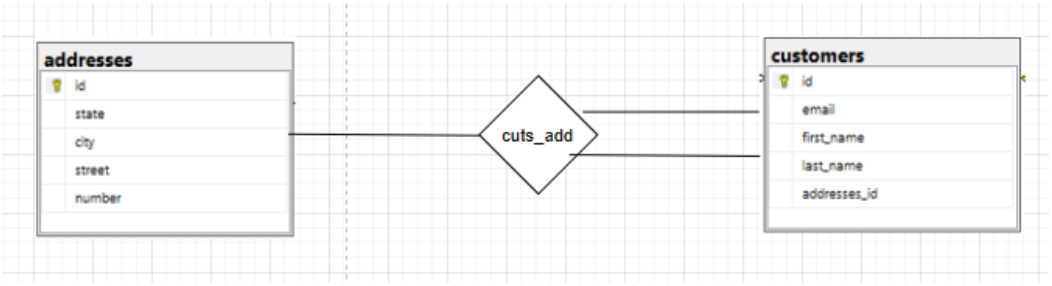


As the diagram shows, premium_customer inherits customer, every premium_customer is actually a customer. They are connected to each other with the id foregein key, they are separated from each other by loyalty points.

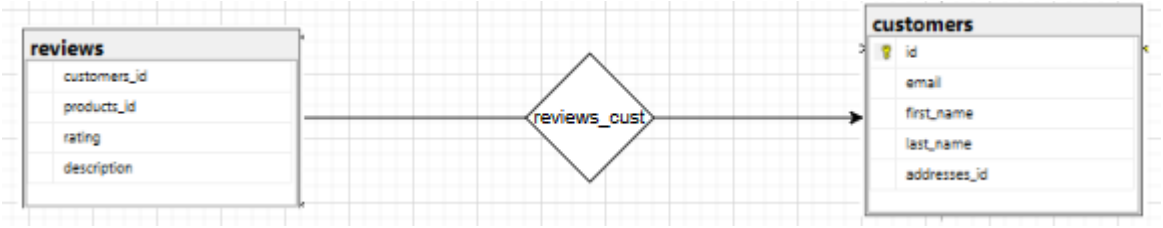
**12) Ternary Relationship**


Ternary relationships are relationships between three entities. There is a tripartite relationship between customers, orders and products. This relationship represents the fact that a customer can place an order for more than one product.

**13) Total and partial completeness constraint**



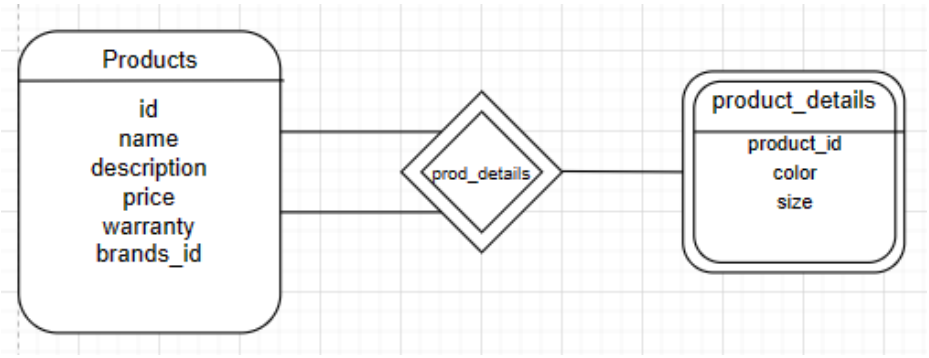

If the **total completeness** constraint is applied to the relationship. Each customer must be associated with at least one address.




Some entities may not participate in any relationship in the relationship set.

Customer's participation in review is **partial**. Because not every customer can review.

**14) Weak Entity**




## Query Examples:

**Creating Table :**

```
CREATE TABLE [dbo].[products] (
    [id]          INT            IDENTITY (1, 1) NOT NULL,
    [name]        VARCHAR (50)   NULL,
    [description] VARCHAR (50)   NULL,
    [price]       DECIMAL (5, 2) NULL,
    [warranty]    INT            NULL,
    [brands_id]   INT            NULL,
    CONSTRAINT [PK_products] PRIMARY KEY CLUSTERED ([id] ASC),
    CONSTRAINT [FK_products_brands] FOREIGN KEY ([brands_id]) REFERENCES [dbo].[brands] ([id])
);
```

**Insert – Update – Delete :**

```
INSERT INTO [dbo].[products] ([name], [description], [price], [warranty], [brands_id])
VALUES ('Product 1', 'Description 1', 9.99, 12, 1);

UPDATE [dbo].[products]
SET [price] = 12.99
WHERE [id] = 1;
```

```sql
DELETE FROM [dbo].[products]
WHERE [id] = 1;
```

**View:**

```sql
CREATE VIEW customer_details_view AS
SELECT customers.id, customers.email, customers.first_name, customers.last_name,
       addresses.state, addresses.city, addresses.street, addresses.number,
       reviews.rating, reviews.description
FROM customers
LEFT JOIN addresses ON customers.addresses_id = addresses.id
LEFT JOIN reviews ON customers.id = reviews.customers_id;
```

```sql
CREATE VIEW product_details_view AS
SELECT products.id, products.name, products.description, products.price, products.warranty,
       categories.name AS category, brands.name AS brand, items.production_date
FROM products
LEFT JOIN categories_has_products ON products.id = categories_has_products.products_id
LEFT JOIN categories ON categories_has_products.categories_id = categories.id
LEFT JOIN brands ON products.brands_id = brands.id
LEFT JOIN items ON products.id = items.products_id;
```

```sql
CREATE VIEW product_details_with_cond_view AS
SELECT products.id, products.name, products.description, products.price, products.warranty,
       categories.name AS category, brands.name AS brand, items.production_date
FROM products
LEFT JOIN categories_has_products ON products.id = categories_has_products.products_id
LEFT JOIN categories ON categories_has_products.categories_id = categories.id
LEFT JOIN brands ON products.brands_id = brands.id
LEFT JOIN items ON products.id = items.products_id
WHERE items.production_date < '2022-01-01';
```

```sql
CREATE VIEW review_details_view AS
SELECT products.id, products.name AS product_name, products.description AS product_description, products.price, products.warranty,
       customers.first_name, customers.last_name,
       reviews.rating, reviews.description AS review_description
FROM products
INNER JOIN reviews ON products.id = reviews.products_id
INNER JOIN customers ON reviews.customers_id = customers.id;
```

**Triggers:**

```
GO
CREATE TRIGGER trg_update_product_price
ON products
AFTER UPDATE
AS
BEGIN
    IF UPDATE(price)
    BEGIN
        DECLARE @oldPrice DECIMAL(5, 2), @newPrice DECIMAL(5, 2);

        SELECT @oldPrice = price FROM deleted;
        SELECT @newPrice = price FROM inserted;

        IF @oldPrice <> @newPrice
        BEGIN
            INSERT INTO price_change_logs (product_id, old_price, new_price, change_date)
            SELECT id, @oldPrice, @newPrice, GETDATE() FROM products WHERE id IN (SELECT id FROM inserted);
        END
    END
END
```

```
GO
CREATE TRIGGER trg_delete_order
ON orders
INSTEAD OF DELETE
AS
BEGIN
    DECLARE @customerId INT;

    SELECT @customerId = customers_id
    FROM deleted;

    DELETE FROM order_items
    WHERE orders_id IN (SELECT id FROM deleted);

    DELETE FROM orders
    WHERE customers_id = @customerId;
END
```

```
GO
CREATE TRIGGER trg_delete_category
ON categories
AFTER DELETE
AS
BEGIN
    DECLARE @categoryId INT;

    SELECT @categoryId = id
    FROM deleted;

    DELETE FROM categories_has_products
    WHERE categories_id = @categoryId;
END
```

**Transactions:**

```sql
BEGIN TRANSACTION;

INSERT INTO [dbo].[customers] ([email], [first_name], [last_name])
VALUES ('example@email.com', 'John', 'Doe');

DECLARE @customerId INT;
SET @customerId = SCOPE_IDENTITY(); -- Eklenen müşterinin ID'sini al


INSERT INTO [dbo].[orders] ([customers_id], [purchase_date])
VALUES (@customerId, GETDATE());

DECLARE @orderId INT;
SET @orderId = SCOPE_IDENTITY(); -- Eklenen siparişin ID'sini al

INSERT INTO [dbo].[order_items] ([orders_id], [items_id])
VALUES (@orderId, 1); -- Satın alınan ürünün ID'sini belirt

COMMIT;
```

With the commit command, the changes are saved permanently, if rollback is used, all changes are undone.