Final Project Report

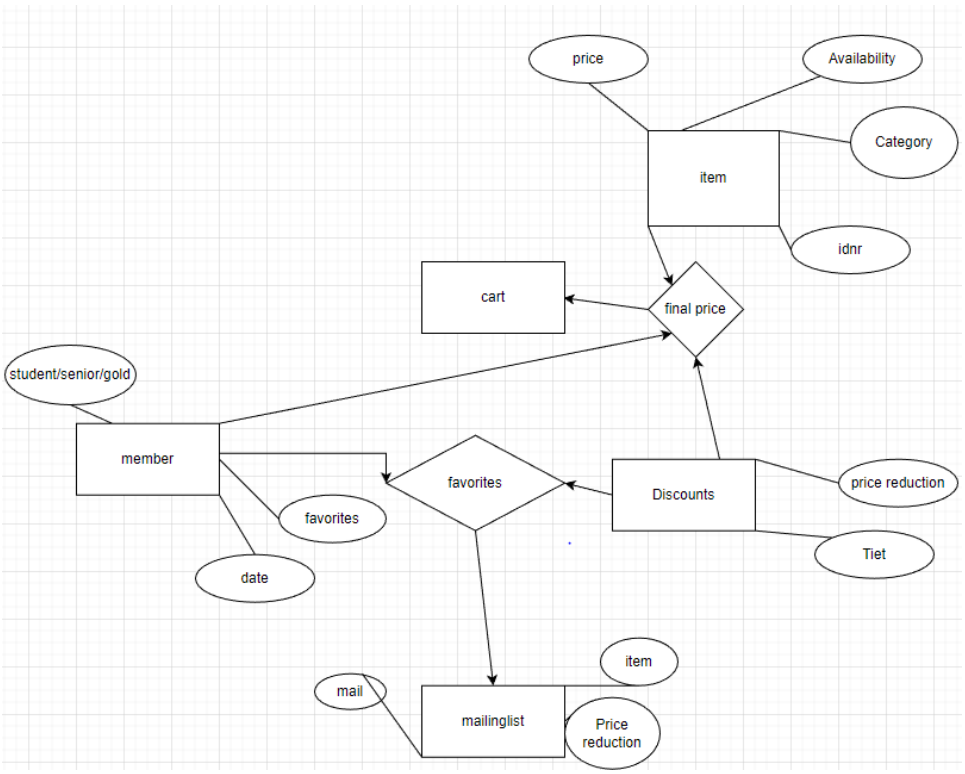
Students: Felix Davidsson – fedv21@student.bth.se
Dennis Calza Wilhelmsson – deca21@student.bth.se

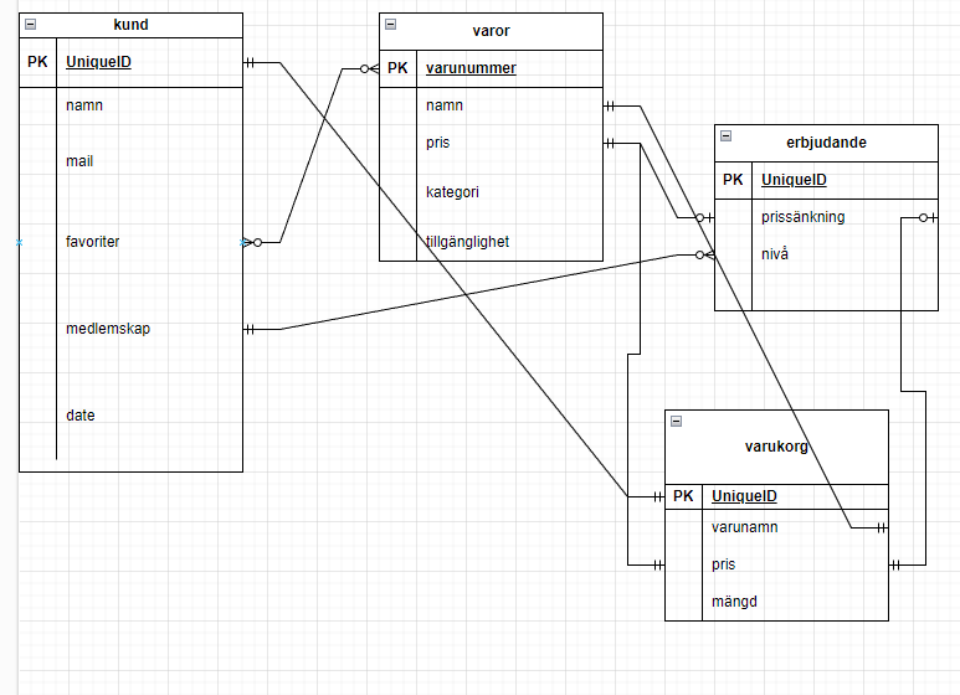
1. Idea

In this project we have created a store database that allows customers to upload their favorite items and categories. The customers can also create a cart and see if it is available and the prices of each item, including discounts. The store administrator can see and change the inventory and discounts. They can also see the mailing list based on the customer's favored items and membership tier. The program will give easy access to all necessary information both for customers and admins, it also provides simple ways to change the favored items. We couldn't find a readily available database from a grocery store, so we decided to generate our own inventory and discounts.

2. Diagrams

Entity relations diagram





One important relation in the schema is the one between customers and discounts. On the first side we have the item/categories that are favorited by a customer and their membership tier. On the other side we have the discounted items id and the required tier. Where these columns are matching will result in the mailing list. Another relationship is between the discount, the inventory and the customer's cart, as the price of an item might change if they're eligible for a discount. The relationship between the inventory and discounts is that when a discount has the same ProductID as an item the price will change for customers with the right tier.

It is important to note that in this diagram the cart is represented in its temporary table form, it is otherwise a string-based list with product id and a uniqueID linking it to a customer.

All tables have some form of index column making them easier to navigate. In cases where choosing the index is not important, they are auto incremented. In the inventory you might want to group certain items together so in that case an index number is chosen manually when inserting new items and in cart it's linked to the customer.

3. queries

1 Trigger: This trigger will insert a row to the cart table when a new customer is added. The new row in cart will have the same uniqueID as the new row in customers and an empty varchar, representing an empty shopping cart. This gives us an ID that can easily be matched with a customer, for example if we want to search for discounts. There is also a corresponding trigger when deleting a customer.

```
CREATE TRIGGER insert_into_cart AFTER INSERT ON customers
FOR EACH ROW
INSERT INTO cart (uniqueID, prodlist) VALUES (NEW.uniqueID, "); " \
WHERE uniqueID = (SELECT uniqueID FROM customers ORDER BY uniqueID DESC LIMIT 1);
```

2 Mailing list: this query will return a table with customer email, discounted item and the decimal value of how much the item costs with the discounts. This query is multirelational between the tables customers and discounts, it uses a left join to match where the customers memberType can be found within the memberType list in discount or if it's available for all tiers. It also uses and/or clauses that check if the item or category is preferred by the customer, to make sure that they only get notified about discounts that they're interested in.

```
SELECT uniqueID, mail, discounts.prodName, discounts.discount
FROM customers
LEFT JOIN discounts
on discounts.memberType like concat('%',customers.memberType,'%') and customers.favoriteProd like concat('%',discounts.prodName,'%')
or discounts.memberType like concat('%',customers.memberType,'%') and customers.favoriteCategory like concat('%',discounts.category,'%')
or discounts.memberType = 'none' and customers.favoriteProd like concat('%',discounts.prodName,'%')
or discounts.memberType = 'none' and customers.favoritecategory like concat('%',discounts.category,'%');
```

3 procedure: the procedure CheckCart takes an uniqueID as argument. To function it uses data from 4 different tables: Cart, customers, inventory and discounts. It extracts the items from prodList in the cart, then checks the price and availability of each item from inventory, it will also check which discounts the customer is eligible for and will calculate a new price. To get the amount of each selected item we use the aggregation function count. The final output of the procedure is the name of the item, the total price of that item, how many of the item is in the cart and quantity in inventory. Finally, we get the total price of the cart. To make it easier to work with the data we created two temporary tables within the procedure, these are dropped before the procedure is done.

```
DELIMITER //


CREATE PROCEDURE CheckCart(IN cartID INT)

BEGIN

   DECLARE prodItem VARCHAR(100);

   DECLARE done INT DEFAULT FALSE;


   DECLARE cursorCartItems CURSOR FOR

      SELECT prodList FROM cart WHERE uniqueID = cartID;


   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;


   CREATE TEMPORARY TABLE IF NOT EXISTS tempItemPrices (
```

```
    item VARCHAR(100),
    price DECIMAL(10, 2),
    available INT
);

OPEN cursorCartItems;

read_loop: LOOP
    FETCH cursorCartItems INTO prodItem;

    IF done THEN
        LEAVE read_loop;
    END IF;

    WHILE LENGTH(prodItem) > 0 DO
        SET @pos := LOCATE(',', prodItem);

        IF @pos = 0 THEN
            SET @item := prodItem;
            SET prodItem := '';
        ELSE
            SET @item := SUBSTRING(prodItem, 1, @pos - 1);
            SET prodItem := SUBSTRING(prodItem, @pos + 1);
        END IF;

        SET @itemPrice := (SELECT price FROM inventory WHERE prodID = TRIM(@item));
        SET @itemAvailable := (SELECT available FROM inventory WHERE prodID = TRIM(@item));

        IF @itemPrice IS NOT NULL THEN
```

```
        SET @memberType := (SELECT memberType FROM customers WHERE uniqueID =
cartID);
        SET @discount := (SELECT discount FROM discounts WHERE prodID = TRIM(@item)
AND FIND_IN_SET(@memberType, memberType));

        IF @discount IS NOT NULL THEN
           SET @itemPrice := @itemPrice * @discount;
        END IF;

        INSERT INTO tempItemPrices (item, price, available) VALUES (TRIM(@item), @itemPrice,
@itemAvailable);

      END IF;

    END WHILE;

  END LOOP;

  CLOSE cursorCartItems;

  create table realCart(
  item VARCHAR(100) default 0,
  price Decimal(10,2) default 0.0,
  quantity INT default 0
);
insert into realCart
select * from  tempItemPrices;

UPDATE realCart t
JOIN (
SELECT item, COUNT(item) AS quantity, price
```

```
FROM tempItemPrices

GROUP BY price, item

) AS sub ON t.item = sub.item

SET t.item = (select prodName from inventory where prodID = t.item), t.quantity = sub.quantity, t.price = t.price * sub.quantity;


INSERT INTO realCart (item, price, quantity) VALUES ('Total', (select sum(price) from tempItemPrices), NULL);


select distinct * from realcart;


   drop table realCart;

   drop table tempItemPrices;

END //


DELIMITER ;
```

4 discount list: gives a summary of all the discounts a certain customer is eligible for. Just like the mailing list it works multirelationaly between customers and discounts, but it does not check if the item is favored by the customer.

```
SELECT discounts.prodName, discounts.discount
FROM customers
LEFT JOIN discounts
ON discounts.memberType LIKE CONCAT('%', customers.memberType, '%')
or discounts.memberType = 'none'
Where uniqueID  = <id>;
```

5 category search, this query will return all the item categories in the inventory and total amount of items in each of them, this makes it easier to know how to categorize new items. The query uses sum to aggregate the quantity of items in each category,

```
Select category, sum(available) as total from inventory group by category;
```

4 Discussion and resources

One issue we encountered was when we wanted to show the result of the cart, we couldn't find a way to manipulate and use the string-based list as we wanted, and we didn't want to create a unique table for each customer as it would take up a lot of space and make the database harder to navigate. The solution

was to create a temporary table where we could access the elements as if they were regular rows, when we're finished with the table it is dropped.

Read the readme file on github for initial installation details.

Additional libraries: Datetime, Tabulate and os

Video link: https://youtu.be/ASfNCbWmsYk

GitHub link: https://github.com/decawi/DBProject---Deca21-Fedv21

Changelog

name	task	date
Felix	Initial setup	05-05
Felix	Base queries/ mailing list and triggers	05-05
Dennis	Github setup	05-24
Dennis	Started work on cart	05-25
Felix	Modified cart procedure	05-26
Dennis/felix	Python implementation	05-26
Dennis	Added support queries	05-27
Dennis/felix	Python implementation	05-28
felix	Final fixes in python/documentation	05-28