CS-A1150 Databases, Spring 2024

Project, Part 2 (UML Model and Relational Data Model)

Christina Zhou, christina.zhou@aalto.fi

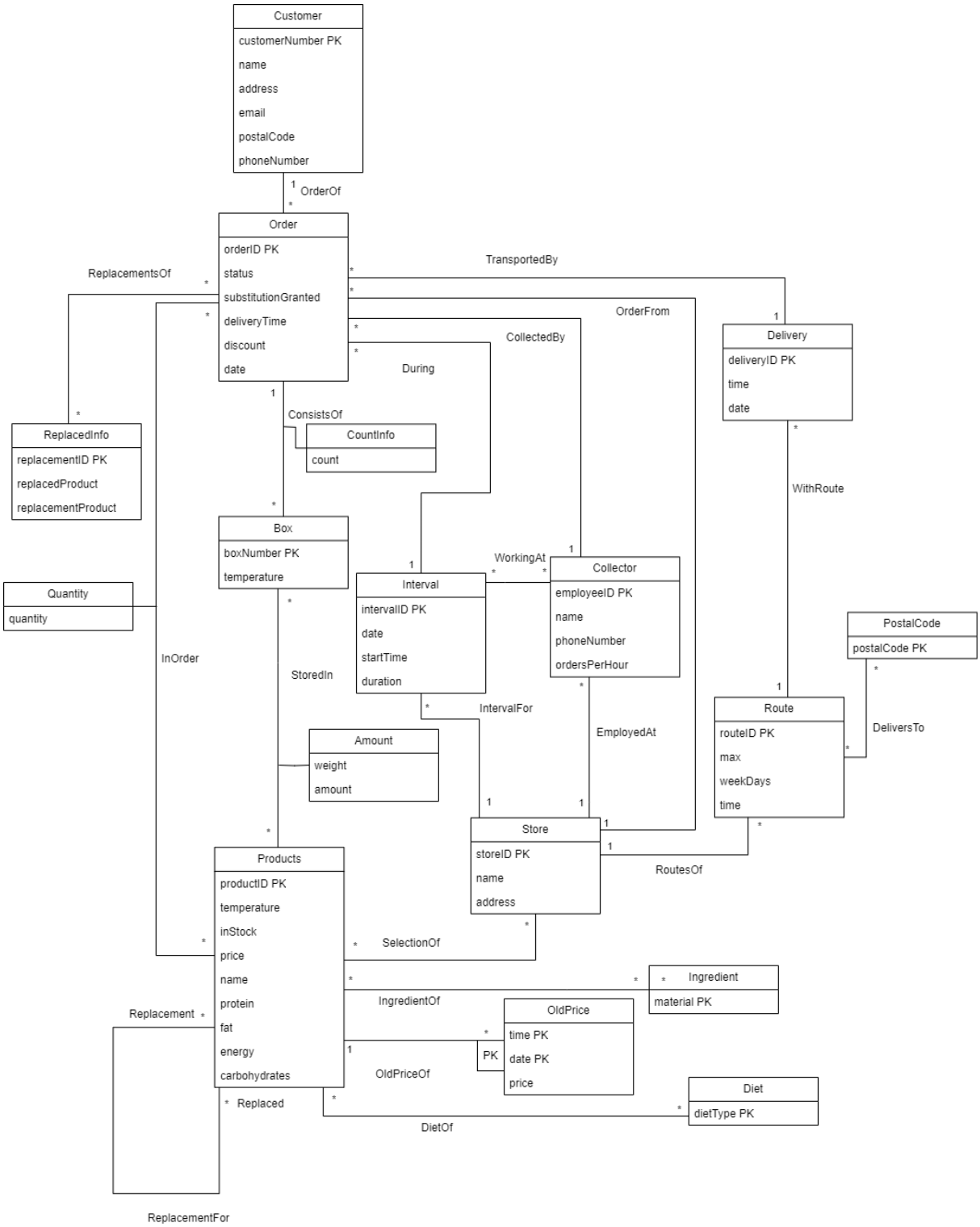
Kanerva Kotilainen, kanerva.kotilainen@aalto.fi

Max Mussalo, max.mussalo@aalto.fi

Contents

PART 1

UML model



Description

The UML of the home delivery contains relations customer, order, postalCode, box, interval, collector, route, store, products, ingredient, old price, diet and replace. Instances of these can be made using these relations. New stores, customers, products, and collectors can be added by adding tuples to their respective relations. Similarly, they can also be deleted by removing tuples.

Customers relation:

- This relation defines a customer's account on the store chain's website. It stores the name, email, address, phone number and postal code.
- A unique customer number is created for each creation of an account. This is set as the primary key.
- The address attribute does not define postalCode since we can have same addresses in different postalCodes.

Store relation:

- Stores are singular locations that customers can make orders to.
- They have names, addresses and a unique storeID.

Order relation:

- When a customer makes an order, a new order instance with an orderID is saved. Other attributes include status, deliveryTime, discount and substutionGranted.
- Orders consist of products and their quantity is defined by the association relation Quantity.
- One can check the status of shipment with the status attribute.
- Old orders are saved for later inspection.
- They can be found by searching for orders with the status shipped.
- If a store runs out of some product, it may replace it with a replacement if the customer grants permission, hence the attribute substutionGranted.
- The total price of the order is calculated using the products that are collected in the orders boxes. This way, we get the price of substitutes or won't charge for items out of stock.
- Some orders may get a discount on the total price.
- Orders can be made in advance for some specific time*

Box relation:

- Orders are made up of boxes and each box has its unique box number, which is the primary key.

- The collectors collect products into these boxes.
- In addition, boxes have types and temperatures. The boxes vary in their inner temperature and store a number of products described by association relation Amount. Different products are better kept in different temperatures.
    - The relation Amount may store the unit of products in grams (weight) or count (amount).

Product relation:

- Each product has its own productID as the primary key.
- Products can be added by first adding them to a box and then adding the box to the order.
- They also have a temperature, price, and name.
- When products are put into boxes, they need to be stored in boxes with matching temperatures.
- Products also contain nutritional values including protein, energy, fat, and carbohydrates in grams.
- As mentioned, products can be replaced by other products if the customer has granted permission. Self-association allows us to refer to another product as the replacement.
- The possible replacements can be found by the Replacement for association and checking if the replacement is in stock.
- Old prices are stored in the system by date and time. This relation is set as a primary key relation, meaning it also uses the product's primary key.

ReplacementInfo relation:

- When a product is out of stock the system logs a new replacement event.
- Each event has a replacementID as its primary key, and information about the substitution and the substituted product.
- If the customer does not want substitutions to be made, the substitute is left empty.
- This way we can keep count of how many products were supposed to be delivered but weren't due to not being in stock.

Ingredient relation:

- The product may consist of multiple ingredients.

Diet relation:

- Information about diets is also included.
- A product can fit multiple diet types.
- One can look for products with certain diets.

Routes relation:

- Routes are used for deliveries and one store can have multiple of them.
- Routes are defined by a routeID, which is also the primary key.
- It also includes the weekdays and time that it operates on, and the maximum number of orders that can be delivered on the route simultaneously.

Collectors relation:

- Stores have employees with a unique employeeID (PK), name, phoneNumber, and ordersPerHour.
- OrdersPerHour are determined by the experience of the employee. This affects the quantity of orders that the employee can prepare in an hour.
- A collector's working hours can be modified in the WorkingAt relation.
- A list of products to be collected can be attained by finding the products in an order, their quantity and storage temperature.

Interval relation:

- Employees work on orders during specific intervals. Intervals are time periods of the day and are identified by their intervalID (PK), date, startTime and duration.
- When orders are scheduled for a specific time, there must be enough space in the route max and the previous interval for workers to gather the order.

Postal code relation:

- Orders are delivered to certain postal codes provided by the customer. The relation PostalCode consists of the PK postal code. Routes can go through multiple postal code areas.

Delivery relation:

- Singular instances of deliveries use specific routes. Multiple orders can be delivered during one delivery. The date and time of delivery are stored, in addition to the deliveryID which is the primary key of this relation.

Relation schema

Customer(customerNumber, name, address, email, phoneNumber, postalCode)

PostalCode(postalCode)

Order(orderID, status, substitutionGranted, deliveryTime, routeID, employeeID, intervalID, customerNumber, count, discount, date)

Route(routeID, max, weekDays, time, storeID)

Collector(employeeID, name, phoneNumber, ordersPerHour, storeID)

Interval(intervalID, date, startTime, duration)

Store(storeID, name, address)

Box(number, temperature, orderID)

Diet(dietType)

Products(productID, temperature, price, name, protein, energy, fat, carbohydrates, inStock)

OldPrice(productID, time, date, price)

ReplacedInfo(replacementID, replacementProduct, replacedProduct)

Ingredient(material)

Delivery(deliveryID, time, date)

Association schema

SelectionOf(productID, storeID)

StoredIn(number, productID, weight, amount)

WorkingAt(employeeID, intervalID)

DeliversTo(routeID, postalCode)

ReplacementOf(replacementID, orderID)

InOrder(orderID, productID, quantity)

ReplacementFor(ReplacementProductID, ReplacedProductID)

DietOf(dietType, ProductID)

IntervalFor(IntervalID, storeID)

OrderFrom(orderID, storeID)

WithRoute(deliveryID, routeID)

TransportedBy(orderID, deliveryID)

Functional dependencies

Relation schema

Customer(customerNumber, name, address, email, phoneNumber, postalCode)

customerNumber -> name address email phoneNumber postalCode

PostalCode(postalCode)

No functional dependencies

Order(orderID, status, substitutionGranted, deliveryTime, routeID, employeeID, intervalID, customerNumber, count, discount, date)

orderID -> status substitutionGranted deliveryTime routeID employeeID intervalID customerNumber count discount date

Route(routeID, max, weekDays, time, storeID)

routeID -> max weekDays time storeID

Collector(employeeID, name, phoneNumber, ordersPerHour, storeID)

employeeID -> name phoneNumber ordersPerHour storeID

Interval(intervalID, date, startTime, duration)

intervalID -> date startTime duration

Store(storeID, name, address)

storeID -> name address

Box(number, temperature, orderID)

number -> temperature orderID

Diet(dietType)

No functional dependencies

Products(productID, temperature, price, name, protein, energy, fat, carbohydrates, inStock)

productID -> temperature price name protein energy fat carbohydrates inStock

OldPrice(productID, time, date, price)

productID time date -> price

ReplacedInfo(replacementID, replacementProduct, replacedProduct)

replacementID -> replacementProduct replacedProduct

Ingredient(material)

No functional dependencies

Delivery(deliveryID, time, date)

deliveryID -> deliveryID time date

Association schema

SelectionOf(productID, storeID)

No functional dependencies

StoredIn(number, productID, weight, amount)

number productID -> weight amount

WorkingAt(employeeID, intervalID)

No functional dependencies

DeliversTo(routeID, postalCode)

No functional dependencies

ReplacementOf(replacementID, orderID)

No functional dependencies

DietOf(dietType, ProductID)

No functional dependencies

IntervalFor(intervalID, storeID)

intervalID -> intervalID storeID

OrderFrom(orderID, storeID)

orderID -> orderID storeID

WithRoute(deliveryID, routeID)

deliveryID -> deliveryID routeID

TransportedBy(orderID, deliveryID)

orderID -> orderID deliveryID

Anomalies

Anomalies do not occur in databases with relations in fourth normal form (4NF). Since all the relations in our database are in 4NF (see next section), our database cannot have any anomalies.

BCNF

All relations are in Boyce-Codd Normal Form (BCNF) because each relations primary keys closures contain every attribute in the relation and the other attributes closures only contain themselves. In other words, all the non-trivial functional dependencies left sides are the relations primary keys and thus our relation is actually in 4NF.

PART 2

Changes made after the submission of the first part

We modified the UML diagram by introducing an IntervalFor association between Store and Interval, OrderFrom association between Order and Store, and we added a Delivery relation to the UML. We also wrote the description and the relation schema for the delivery relation, and the association schema for IntervalFor, WithRoute, and TransportedBy, as well as the functional dependencies for Delivery, IntervalFor, WithRoute, and TransportedBy. In addition, we added date to Order.

Defining the database schema in SQL

```
CREATE TABLE PostalCode(
        postalCode TEXT NOT NULL PRIMARY KEY
);

CREATE TABLE Customer(
        customerNumber TEXT NOT NULL,
        name TEXT,
        address TEXT,
        email TEXT,
        phoneNumber TEXT,
        postalCode TEXT REFERENCES PostalCode(postalCode),
        PRIMARY KEY (customerNumber)
);

CREATE TABLE Store(
        storeID TEXT NOT NULL PRIMARY KEY,
        name,
        address TEXT
);
```

```
CREATE TABLE Route(
        routeID TEXT NOT NULL PRIMARY KEY,
        max INTEGER NOT NULL,
        weekDays TEXT CHECK (weekDays IN ('monday', 'tuesday', 'wednesday', 'thursday',
        'friday', 'saturday', 'sunday')),
        time TEXT NOT NULL,
        storeID TEXT NOT NULL,
        FOREIGN KEY (storeID) REFERENCES Store(storeID)
);

CREATE TABLE Delivery(
        deliveryID TEXT NOT NULL PRIMARY KEY,
        time TEXT NOT NULL,
        date TEXT NOT NULL,
        routeID TEXT REFERENCES Route(routeID)
);
```

The word Order is a keyword in SQLiteStudio so we decided to rename it as Orderr.

```
CREATE TABLE Orderr(
        orderID TEXT NOT NULL,
        status TEXT CHECK (status IN ('shipped', 'in preparation')),
        substitutionGranted INT CHECK (substitutionGranted IN (0, 1)),
        deliveryTime TEXT,
        routeID TEXT NOT NULL,
        employeeID TEXT NOT NULL,
        intervalID TEXT,
        customerNumber TEXT NOT NULL,
```

```
    count INT,
    discount INT CHECK (discount >= 0 AND discount <= 100),
    date TEXT NOT NULL,
    deliveryID TEXT REFERENCES Delivery(deliveryID),
    storeID TEXT REFERENCES Store(storeID),
    PRIMARY KEY (orderID)
);

CREATE TABLE Collector(
    employeeID TEXT NOT NULL PRIMARY KEY,
    name TEXT NOT NULL,
    phoneNumber TEXT NOT NULL,
    ordersPerHour INTEGER NOT NULL,
    storeID TEXT REFERENCES Store(storeID)
);

CREATE TABLE Interval(
    intervalID TEXT NOT NULL PRIMARY KEY,
    date TEXT NOT NULL,
    startTime TEXT NOT NULL,
    duration TEXT NOT NULL,
    storeID TEXT NOT NULL REFERENCES Store(storeID)
);

CREATE TABLE Box(
    number TEXT NOT NULL PRIMARY KEY,
    temperature REAL,
    orderID TEXT NOT NULL
```

```
);

CREATE TABLE Diet(
        dietType TEXT NOT NULL PRIMARY KEY
);

CREATE TABLE Products(
        productID TEXT NOT NULL PRIMARY KEY,
        temperature REAL,
        price REAL CHECK (price > 0.0),
        name TEXT,
        protein REAL,
        energy REAL,
        fat REAL,
        carbohydrates REAL,
        inStock INT CHECK (inStock IN (0, 1))
);

CREATE TABLE OldPrice(
        productID TEXT REFERENCES Products(productID),
        time TEXT NOT NULL,
        date TEXT NOT NULL,
        price REAL NOT NULL,
        PRIMARY KEY (productID, time, date)
);

CREATE TABLE ReplacedInfo(
        replacementID TEXT NOT NULL PRIMARY KEY,
```

```
    replacementProduct NOT NULL,
    replacedProduct NOT NULL
);

CREATE TABLE Ingredient(
    material TEXT PRIMARY KEY
);

CREATE TABLE SelectionOf(
    productID REFERENCES Products(productID),
    storeID REFERENCES Store(storeID),
    PRIMARY KEY (productID, storeID)
);

CREATE TABLE StoredIn(
    number TEXT REFERENCES Box(number),
    productID TEXT REFERENCES Products(productID),
    weight REAL,
    amount INT,
    material TEXT,
    PRIMARY KEY (number, productID)
);

CREATE TABLE WorkingAt(
    employeeID TEXT REFERENCES Collector(employeeID),
    intervalID TEXT REFERENCES Interval(intervalID),
    PRIMARY KEY (employeeID, intervalID)
);
```

```
CREATE TABLE DeliversTo(
        routeID TEXT REFERENCES Route(routeID),
        postalCode TEXT REFERENCES PostalCode(postalCode),
        PRIMARY KEY (routeID, postalCode)
);

CREATE TABLE ReplacementOf(
        replacementID TEXT REFERENCES ReplacedInfo(replacementID),
        orderID TEXT REFERENCES Orderr(orderID),
        PRIMARY KEY (replacementID, orderID)
);

CREATE TABLE InOrder(
        orderID TEXT REFERENCES Orderr(orderID),
        productID TEXT REFERENCES Products(productID),
        quantity INT NOT NULL,
        PRIMARY KEY (orderID, productID)
);

CREATE TABLE ReplacementFor(
        ReplacementProductID TEXT REFERENCES Products(productID),
        ReplacedProductID TEXT REFERENCES Products(productID),
        PRIMARY KEY (ReplacementProductID, ReplacedProductID)
);

CREATE TABLE DietOf(
        dietType TEXT REFERENCES Diet(dietType),
```

```
    productID TEXT REFERENCES Products(productID),
    PRIMARY KEY (dietType, productID)
);
```

Explanation for the attribute types:

ID, times, dates, and numbers are labeled as text. These are all either identifiers or times/dates. We don't need to do any calculations with identifiers, so they don't need to be integers or real numbers. SQLite supports many different attribute types for times/dates, and we chose to use text.

Quantities, weights, and counts are real or int. We may need to use these in calculations and thus real and int are the best choices.

Temperature and prices are real, since they don't necessarily have to be whole numbers.

Discount is written as Int and refers to the percentage of discount. In our project percentages are always integers, which is why we don't need to use real.

Truth values are Int because SQLite doesn't support Boolean values. 0 is false and 1 is true.

Typical queries and their outputs

Check total price of the order with orderID 'order001'. This could be for displaying the total price for the customer.

The query uses the natural join operator to find the order with orderID of 'order001' and what products each of those contain. Then the price of the product is multiplied by the quantity of that product, and finally summed together using the sum aggregate function.

This query requires the initialization, updating of and insertion of Orderr, inOrder and Customer. These also require the initialization of Deliveries, stores, and products to function.

```
SELECT SUM(price*quantity)
FROM Orderr NATURAL JOIN (InOrder NATURAL JOIN Products)
WHERE orderID = 'order001';
```

	SUM(price*quantity)
1	14.75

[13:36:31] Query finished in 0.000 second(s).

Checking old orders for complaints with orderID 'order001':

The query finds all orders with orderID ' order001'. Then the customer/employee can check the complaint.

This requires the customer's order to be inserted into orders and the order to be updated as it is shipped, collected, replacement is granted etc.

```
SELECT *

FROM Orderr

WHERE orderID = 'OR001';
```

	orderID	status	substitutior	deliveryTim	routeID	employeeII	intervalID	customerNi	count	discount	date	deliveryID	storeID
1	order001	in preparation	1	NULL	route001	emp001	NULL	Cust001	1	0	2024-05-05	delivery001	store001

[13:37:51] Query finished in 0.001 second(s).

Checking working intervals for salary calculations for employee with employeeID 'emp001':

The query finds which intervals employee 'emp001' has worked in during the last 30 days. It then calculates the sum of the duration of the intervals. The employer can then use this info to determine the employee's salary.

This requires the employee to be inserted into collector, the right working intervals to be inserted and the times when the collector is working to be correct.

```
SELECT SUM(duration)

FROM Interval NATURAL JOIN (WorkingAt NATURAL JOIN Collector )

WHERE employeeID = 'emp001' AND date >= DATE('now', '-30 day');
```

	SUM(duration)
1	2

[13:39:20] Query finished in 0.001 second(s).

Price trends over time of product with productID 'prod001':

Extracts the time, date and price of a certain product and displays it on a table. A graph can later be acquired using the table to further visualize the data.

This requires the initialization, insertion and updating of products and old prices.

```
SELECT time, date, price

FROM OldPrice

WHERE productID = 'prod001';
```

	time	date	price
1	10:00	2024-05-01	3.49
2	21:00	2024-05-01	3.19
3	5:00	2024-05-01	3.89
4	9:00	2024-05-01	3.6

[13:47:57] Query finished in 0.001 second(s).

Find the most popular product in category 'Milk' and suggest it to the customer:

This query checks all milk products, which have been ordered and groups them by their productid in descending order. It then selects the first productID.

It requires the products to be inserted into Products and orders inserted into Orderr. It also requires the InOrder to be updated correctly.

```
SELECT productID

FROM  (SELECT productID, COUNT(quantity) AS C

        FROM Orderr NATURAL JOIN (InOrder NATURAL JOIN Products)

        WHERE name = 'Milk'

        GROUP BY productID

        ORDER BY C DESC

        LIMIT 1);
```

	productID
1	prod002

[13:39:34] Query finished in 0.000 second(s).

Checking popularity of different routes from the past month for store with storeID ' store001':

This query checks all the tuples during the last month in Orderr where the storeID is 'store001'. It then groups these based on the routeID. The tuples are then counted for each route. The popularity can then be deduced from these.

This requires the customer's order to be inserted into orders. The store where the order is collected and the route it takes need to be updated.

```
SELECT routeID, COUNT(*)

FROM Orderr

WHERE storeID = 'store001' AND date >= DATE('now', '-30 day')

GROUP BY routeID;
```

	routeID	COUNT(*)
1	route001	5
2	route002	5

[13:40:01] Query finished in 0.000 second(s).

Checking popularity of store specific online orders during the past month and whether it sustainable to upkeep them:

The query finds all orders of each store from the last 30 days and groups those orders. Then it counts the quantity of orders for each store and displays it as a table.

The initialization, updating of and insertion of stores and orders. Their initialization in turn require deliveries, postal codes and routes to be initialized.

```
SELECT storeID, Count(orderID)

FROM Store NATURAL JOIN Orderr

WHERE date >= DATE('now', '-30 day')

GROUP BY storeID;
```

	storeID	Count(orderID)
1	store001	10
2	store002	10

[13:40:29] Query finished in 0.000 second(s).

Checking which delivery times are busy:

Customers can place pick up times for their orders and the query checks what time has the most orders to be made at. It picks the orders with an initialized deliveryTime that are still yet to be prepared and displays how many orders different times have.

This only requires the initialization and insertion of orders which then also requires deliveries, and stores to be initialized.

SELECT deliveryTime, COUNT(*)

FROM Orderr

WHERE status = 'in preparation' AND deliveryTime NOT NULL

GROUP BY deliveryTime;

	deliveryTime	COUNT(*)
1	17:00	5

[13:53:46] Query finished in 0.000 second(s).

Check name of products in stock for store 'store001':

Check the selection of store with storeID 'store001'. It then checks all the products with inStock value 1(true). Then it selects the products productID and name.

This requires all the products in the 'store001' to be inserted into Products and their inStock value to be updated. It also requires all the products to be inserted into SelectionOf.

SELECT productID, name

FROM Products NATURAL JOIN SelectionOf

WHERE inStock = 1 AND storeID = 'store001';

	productID	name
1	prod001	Milk
2	prod002	Milk
3	prod003	Bread
4	prod004	Bread
5	prod005	Eggs
6	prod006	Eggs
7	prod007	Soda
8	prod008	Orange Juice
9	prod009	Frozen Pizza
10	prod010	Cheese

[13:41:13] Query finished in 0.001 second(s).

Which products the store 'store001' offers that is in category 'milk' is 'Lactose-Free' and in stock:

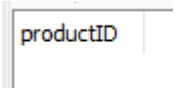
Products of the type milk and diet of lactose-free are found after which their availability is checked for a specific store. The query returns all the products that satisfy these requirements.

This requires all the products in the 'store001' to be inserted into Products and their inStock value to be updated. It also requires all the products to be inserted into SelectionOf and for products to be assigned their corresponding dietTypes using DietOf.

SELECT productID

FROM (Products NATURAL JOIN DietOf) NATURAL JOIN SelectionOf

WHERE name = 'Milk' AND dietType = 'Lactose-Free' AND inStock = 1 AND storeID = 'store001';



productID

[13:41:28] Query finished in 0.000 second(s).

Which products are in category x, for example 'milk':

This selects all products with name milk.

It only requires the products to be updated to the Products table.

SELECT *

FROM Products

WHERE name = 'Milk';

	productID	temperatur	price	name	protein	energy	fat	carbohydra	inStock
1	prod001	4	3.19	Milk	3.2	42	2	30	1
2	prod002	4	2.79	Milk	2.8	38	1.5	28	1

[13:42:19] Query finished in 0.001 second(s).

How many products are in category 'Bread':

This selects all products with name bread and then counts the number of results.

It only requires the products to be updated to the Products table.

SELECT COUNT(*)

FROM Products

WHERE name = 'Bread';

	COUNT(*)
1	2

[13:42:34] Query finished in 0.002 second(s).

Find all products replaced in order 'order001':

This query finds all products that were replaced in an order.

The proper updating of replacement information must be made everytime a replacement happens.

SELECT replacedProduct

FROM ReplacedInfo NATURAL JOIN ReplacementOf

WHERE orderID = 'order001';

	replacedProduct
1	prod001

[13:42:47] Query finished in 0.001 second(s).

Number of employees in a store with storeID 'store001':

This query counts the number of employees working at a specific store, identified by the storeID 'store001', by joining the Collector table with the Store table on the basis of storeID.

SELECT COUNT(*)

FROM Collector AS C JOIN Store AS S ON C.storeID = S.storeID

WHERE C.storeID = 'store001';

	COUNT(*)
1	2

[13:43:01] Query finished in 0.000 second(s).

Stores containing products with dietType 'Lactose-Free':

This first selects all products, dietTypes and in which stores they are sold. It then requires them to have dietType 'Lactose-Free' and returns the products ID name and where it is sold.

This requires the products to be inserted into products. It also requires their dietTypes to be inserted into DietOf and into SelectionOf. The selectionOf of needs to be updated as new stores include the product in their selections.

SELECT productID, storeID, name

FROM (DietOf NATURAL JOIN SelectionOf) NATURAL JOIN Products

WHERE dietType = 'Lactose-Free';

	productID	storeID	name	storeID:1
1	prod001	store001	Milk	store001

[13:43:13] Query finished in 0.001 second(s).

Indexing

CREATE INDEX EmployeeIndex1 ON Collector(name);

CREATE INDEX EmployeeIndex2 ON Collector(employeeID);

Indexes for collectors names and IDs are useful since employers might search for employees based on names and many things such as salaries are calculated with employeeIDs. These also narrow the selection a lot.

CREATE INDEX ProductIndex1 ON Products(name);

CREATE INDEX ProductIndex2 ON Products(productID);

Store workers might search for genres of products based on name e.g 'maito' and for example the price is calculated based on productID. These also narrow the selection a lot.

CREATE INDEX OrderIndex ON Orderr(orderID);

These can be used when checking orders. OrderID is unique so this narrows the number of orders down to one.

CREATE INDEX RouteIndex ON Route(routeID);

This speed up the process calculating routes for orders deliveries.

CREATE INDEX CustomerIndex ON Customer(customerNumber);

Speeds up the process of finding customers address for the order.

View definition

The products a customer has ordered

This might make it easier for a user interface to deduce which products a certain customer has ordered.

CREATE VIEW CustomersProducts AS

SELECT Products.productID, name, customerNumber

FROM Products NATURAL JOIN (InOrder NATURAL JOIN (Orderr NATURAL JOIN OrderOf))

WHERE Orderr.status = 'in preparation';