CS Store promotions

There are 100 points you can get in this assignment.

The assignment is to create a supermarket database for our CS Store example. The database will be called CS_Store. Most of the tables are what you would expect from the videos on SQL (with a similar, but not always identical set of attributes). The twist compared to the slides (otherwise, if you could just follow the slides without thinking, it would be a bit too easy) is that we will also want to support logisitics. I.e. moving wares around the different shops.

Question 2-4 and 6 will each give 15 points. 5 point is given for getting the right output on the test data described at the end – each question will specify what this right output is and you will also be able to see it when you upload your solution to CodeGrade (since you can upload any number of times before the deadline, I would suggest using it to test your solution against the test data) – and another 10 points is given for getting the right output on another data set. The latter set is kept hidden to avoid you hardcoding the right output in the queries. It will follow the same form as the test data at the end though and unless you hardcode your solution for the test data, it is very likely that if you get points for one part you will get points for the other. Question 5 is similar but will award 19 points for getting the right output on the hidden dataset (still 5 point for the test data at the end), because Question 5 is quite tricky. Question 6 will be hard to do without doing question 5 though.

As an aside, each question from 2-6 asks you to create a view, that you should be sure only has each line once (i.e. use DISTINCT! – that said, GROUP BY will automatically give you DISTINCT, even without specifying it, because of how GROUP BY works) and specifies how to sort it – it is necessary for how we grade it. Do make sure you do the latter, since it would be sad for you to lose points for not doing something relatively easy like that! Doing it this way ensures that each question has a unique correct output of the queries on both the test data at the end as well as the hidden set of test data (but there are multiple ways of doing all the queries) and grading will consists of checking that you get the right outputs (there are too many of you to do this by hand and it would lead to errors in grading if I did).

You may use as many views as you wish to solve each question (well, except question 1, since it would not be helpful).

Deadline and feedback

The deadline of the assignment is **Friday the 29th of October**. General feedback for the assignment will be given Monday the 15th of November. The relative long period between those is because you can get extensions (you need to fill out a form online for that however) to hand in, but these can't be longer than 2 weeks or to whenever I give feedback, whichever come first. There are very many students (~500) on this course and some will therefore, statistically speaking, have very good reasons why they need to delay their assignment as much as possible. To be as nice as I can to them, I will therefore first give feedback a bit over 2 weeks after the deadline… If you have specific concerns about your grade or similar for the first assignment, then, after the general feedback has been released, I will answer questions about your solution and why you got the grade you got in the Q&A sessions.

Format

The assignment should be done in .sql format (i.e. the output format from MySQLs workbench) – it is really just a basic text file with the SQL commands written in it and you could do it by writing the file directly, if you wish – I would suggest not to, but you could.

The name of the file should be cs_store.sql: You can hand in precisely 1 file and it must have precisely that name (you can hand in multiple times though until the deadline, but only the most recent version count).

And each line should contain only the following:

1. CREATE TABLE statements for question 1 (8 in total)
2. CREATE VIEW statements for questions 2-6 (the number of views depends on how you solve the questions and how many you solve, if not all). Note, you may use any positive number of views to solve each question, but each questions specified view should have the properties requested.
3. SQL comments, i.e. the part of lines after "-- ", i.e. double - followed by space. You do not need to make any, but may do so if you wish.

Make sure that you can run the full file through MySQL when using the CS_Store database (starting with an empty CS_Store database) and after having done so, the CS_Store database should contain the tables and views required from the questions you solved (and perhaps some more views if you feel it would be convenient). This means that **you should remove any statement that causes errors before handing in the assignment,** because MySQL stops when it encounters an error (meaning that the last statements are not executed)! If you do not, you risk getting a far lower grade than otherwise (because the part of your hand-in after the first error will not be graded)...

You can submit any number of times before the deadline: We are using CodeGrade for checking these things and whenever you submit, you will see whether your file works for the public dataset. I suggest using it…

Do *not* do the following: Any of the following should ***not*** be done:

- End by removing the database (i.e. DROP DATABASE CS_Store; or similar). It would be the same as handing in an empty file.
- Create comments like "------------". MySQL workbench will accept it, but the command line version of MySQL does not, which is what is used to check...
- Swap the columns in the created tables. Since the insert command does not state which columns they insert into, you will put the information in the wrong column and then get hard to understand issues when you solve the questions.

Question 1) (worth 16 points – 2 point for each table)

Make the following set of tables.

- **Customers**(birth_day, first_name, last_name, <u>c_id</u>)
- **Employees**(birth_day, first_name, last_name, <u>e_id</u>)
- **Locations**(address, <u>l_id</u>)
- **Transactions**(e_id*, c_id*, l_id*, date, <u>t_id</u>)
- **Items**(price_for_each, <u>name</u>)
- **ItemsBroughtIntoShop**(name*, l_id*, amount,date)
- **MovementOfItems**(name*, from_l_id*, to_l_id*, amount, date)
- **ItemsInTransactions**(name*, t_id*, amount)

Only use data types in the following list: INT, VARCHAR(20), DATE. Each underlined attribute should be the primary key for the table and each attribute with * should have a foreign key to the table with a primary key of the same name (to be precise, the primary keys are the last attribute in each of the first five tables. The last two tables does not have primary keys), e.g. if the tables were R(a,<u>b</u>) and S(b*,<u>c</u>), b in R and c in S should be the primary keys and b in S should reference b in R as a foreign key. The exception is that from_l_id and to_l_id in MovementOfItems should each (on their own) reference l_id in Locations as a foreign key. To be very clear, each primary key and each foreign key (we reference from) consists of one attribute.

Instead of specifying the datatypes explicitly, ensure that the test data defined at the end gets inserted correctly (it seems very likely that you would also guess the same datatypes as these suggests – well, after noting that strings should be VARCHAR(20)) and use DATE if all entries are dates - recall that you should only use data types in the list: INT, VARCHAR(20), DATE. If you follow all of these requirements, each attribute should have a clear, unique datatype (which happens to likely be what you would guess it to be). As an aside, the prices are measured in pennies (and not directly pounds), to avoid precision issues with floating point numbers.

Question 2) (worth 15 points – 5 point for getting the right output on the test data and another 10 for the hidden data – see the beginning for more detail!)

We are considering giving Denise Davies a raise but want to check how many transactions she has made in September 2021 first. More precisely, you are asked to create a view **DeniseTransactions** with number_of_transactions which should be how many transactions was done by Denise in September 2021 (you may assume she did some and that she is the only employee with that name). As the output is meant to be a single number sorting matters little (still, if you for some reason have chosen a way that generates multiple lines of the same answer, do use DISTINCT).

Note that in the test data, Denise did 3 transactions, of which 2 where in September 2021.

The view should be such that the output of

SELECT * FROM **DeniseTransactions**;

when run on the CS_Store database (after inserting the test data at the end) should be:

number_of_transactions
2

Question 3)

(worth 15 points – 5 point for getting the right output on the test data and another 10 for the hidden data – see the beginning for more detail!)

It was found out that someone in the shop with location id 1 on the 2021-9-07 had COVID-19 and we are supposed to reach out to the involved people. Find the (distinct) people (i.e. both employees and customers) in the shop 2021-9-07. People are assumed to be in the shop if and only if they did a transaction that day. More precisely, you are asked to create a view **PeopleInShop** (use DISTINCT), with birth_day, first_name, last_name of the involved people, sorted by birth_day ascending. HINT: You should likely use UNION and you would need to use a sub-query here to do the sorting (that or use an intermediate view).

Note that in the test data, 3 transactions was done on that date, but 1 was in another shop and the two last had the customer in common.

The view should be such that the output of

SELECT * FROM **PeopleInShop**;

when run on the CS_Store database (after inserting the test data at the end) should be:

birth_day	first_name	last_name
1990-07-03	Anita	Taylor
1991-02-19	Finn	Wilson
1998-08-12	Denise	Davies

Question 4)

(worth 15 points – 5 point for getting the right output on the test data and another for the hidden data – see the beginning for more detail!)

Find the value of each distinct transaction made. More precisely, you are asked to create a view **TransactionValue**, with t_id, value with the value being the sum of the values of the items (i.e. price for each of items times the amount of that item in the transaction) in the transaction, sorted by t_id ascending.

As an example, consider transaction 1 in the test data. It involves (according to **ItemsInTransactions**) 5 Garlic, 8 Bread, 1 Chicken and 1 Rice. The price of (from **Items**) of Garlic is 25, Bread is 200, Chicken is 450 and Rice is 200.

$$5 \cdot 25 + 8 \cdot 200 + 1 \cdot 450 + 1 \cdot 200 = 2125$$

The view should be such that the output of

SELECT * FROM **TransactionValue**;

when run on the CS_Store database (after inserting the test data at the end) should be:

t_id	value
1	2375
2	2750
3	650
4	12175
5	2450
6	5300

Question 5)

(worth 24 points – 5 point for getting the right output on the test data and another 19 points for the hidden data – see the beginning for more detail!)

(This question is really just meant to help guide you to a solution to Question 6). For each distinct type of item, location and date (among those mentioned as the date in transactions or in movement of items), find the amount of that item on that location at that time, if it is non-zero (i.e. if the amount of something is 0 at a location at some time, it should not be included in the output). Note: This amount may be negative, indicating that some issue has occurred (like, someone forgot to insert that some of the items had been moved in from elsewhere or there were more initially or similar): The goal of question 6 is then later to find if any such issue ever occurred, using this.

More precisely, create a view **ItemsOnDateAndLocation** that returns name, l_id, date and amount, where amount should be the amount of the item with that name on that date (among the dates explicitly mentioned in **MovementOfItems**, **Transactions** and **ItemsBroughtIntoShop**) and location, if that amount is non-zero (if it is zero, it, as mentioned above, should not be returned). Sort it by name, l_id and date. Note, there is an amount of wares based on **ItemsBroughtIntoShop** (from the date given there) and some are moved to the location as described by **MovementOfItems**. On the other hand, some items are sold as given by the **itemsInTransactions**/**Transactions** and some are moved from this location (again in **MovementOfItems**). You are meant to take all that into account when finding how much of each item is at each location at a given point in time. You may assume that all amounts are non-negative, but I imagine it won't matter that much.

To make it easier, you may assume that there no issues based on the exact time to move items. In particular, the following kind of issue will *not* be present in the data: Say we have 2 locations, neither of which have any bread at the start of some day. We then move 1 bread from one of the locations to the other and 1 bread from the other location to the first on that date. This can't be done for obvious reasons, would be tricky to check for and you may assume that such issues do not arise.

HINTS: Try to make intermediate/extra views, one for how the item count change on specific dates and one for the mentioned dates. You can then make a combined answer from those.

The expected output is given at the end (but before the test data), since it is long and would distract.

Question 6)

(worth 15 points – 5 point for getting the right output on the test data and another 10 for the hidden data – see the beginning for more detail!)

We want to know for each location whether it always had a feasible (i.e. non-negative) amount of each item. More precisely, create a view **FeasibleLocations** with l_id, feasible, where feasible should be 1 if the amount at all times in l_id for each item (according to **ItemsOnDateAndLocation**) were non-negative and otherwise be 0. Sort it by l_id ascending.

HINT: Use **ItemsOnDateAndLocation** in a subquery with Locations on the outside.

The view should be such that the output of

SELECT * FROM **FeasibleLocations**;

when run on the CS_Store database (after inserting the test data at the end) should be:

l_id	feasible
1	1
2	0
3	0

Question 5 output)

The view you create in question 5 should be such that the output of

SELECT * FROM **ItemsOnDateAndLocation**;

when run on the CS_Store database (after inserting the test data at the end) should be:

name	l_id	date	amount
Banana	1	2021-01-01	5
Banana	1	2021-02-01	5
Banana	1	2021-03-01	5
Banana	1	2021-04-01	5
Banana	1	2021-05-01	5
Banana	1	2021-06-01	5
Banana	1	2021-07-01	14
Banana	1	2021-08-01	14
Banana	1	2021-08-09	14
Banana	1	2021-08-14	14
Banana	1	2021-09-01	10
Banana	1	2021-09-07	10
Banana	1	2021-09-23	10
Banana	1	2021-10-01	10
Banana	2	2021-01-01	7
Banana	2	2021-02-01	7
Banana	2	2021-03-01	7
Banana	2	2021-04-01	7
Banana	2	2021-05-01	7
Banana	2	2021-06-01	2

Banana	2	2021-07-01	2
Banana	2	2021-08-01	2
Banana	2	2021-08-09	2
Banana	2	2021-08-14	-1
Banana	2	2021-09-01	3
Banana	2	2021-09-07	3
Banana	2	2021-09-23	3
Banana	2	2021-10-01	3
Banana	3	2021-01-01	6
Banana	3	2021-02-01	6
Banana	3	2021-03-01	6
Banana	3	2021-04-01	6
Banana	3	2021-05-01	6
Banana	3	2021-06-01	11
Banana	3	2021-07-01	2
Banana	3	2021-08-01	2
Banana	3	2021-08-09	2
Banana	3	2021-08-14	2
Banana	3	2021-09-01	2
Banana	3	2021-09-07	2
Banana	3	2021-09-23	2
Banana	3	2021-10-01	2
Bread	1	2021-01-01	16
Bread	1	2021-02-01	16
Bread	1	2021-03-01	16
Bread	1	2021-04-01	16
Bread	1	2021-05-01	16
Bread	1	2021-06-01	16
Bread	1	2021-07-01	16
Bread	1	2021-08-01	16
Bread	1	2021-08-09	8
Bread	1	2021-08-14	8
Bread	1	2021-09-01	8
Bread	1	2021-09-07	8
Bread	1	2021-09-23	8
Bread	1	2021-10-01	8
Bread	3	2021-01-01	5
Bread	3	2021-02-01	5
Bread	3	2021-03-01	5
Bread	3	2021-04-01	5
Bread	3	2021-05-01	5
Bread	3	2021-06-01	5
Bread	3	2021-07-01	5
Bread	3	2021-08-01	5

Bread	3	2021-08-09	5
Bread	3	2021-08-14	5
Bread	3	2021-09-01	5
Bread	3	2021-09-07	5
Bread	3	2021-09-23	5
Bread	3	2021-10-01	5
Chicken	1	2021-01-01	2
Chicken	1	2021-02-01	2
Chicken	1	2021-03-01	2
Chicken	1	2021-04-01	2
Chicken	1	2021-05-01	2
Chicken	1	2021-06-01	2
Chicken	1	2021-07-01	2
Chicken	1	2021-08-01	2
Chicken	1	2021-08-09	1
Chicken	1	2021-08-14	1
Chicken	1	2021-09-01	1
Chicken	2	2021-01-01	19
Chicken	2	2021-02-01	19
Chicken	2	2021-03-01	19
Chicken	2	2021-04-01	19
Chicken	2	2021-05-01	19
Chicken	2	2021-06-01	19
Chicken	2	2021-07-01	19
Chicken	2	2021-08-01	19
Chicken	2	2021-08-09	19
Chicken	2	2021-08-14	14
Chicken	2	2021-09-01	14
Chicken	2	2021-09-07	14
Chicken	2	2021-09-23	14
Chicken	2	2021-10-01	14
Garlic	1	2021-01-01	1
Garlic	1	2021-02-01	1
Garlic	1	2021-03-01	1
Garlic	1	2021-04-01	1
Garlic	1	2021-05-01	1
Garlic	1	2021-06-01	21
Garlic	1	2021-07-01	21
Garlic	1	2021-08-01	21
Garlic	1	2021-08-09	16
Garlic	1	2021-08-14	16
Garlic	1	2021-09-01	16
Garlic	1	2021-09-07	9
Garlic	1	2021-09-23	9

Garlic	1	2021-10-01	9
Garlic	2	2021-01-01	1
Garlic	2	2021-02-01	1
Garlic	2	2021-03-01	1
Garlic	2	2021-04-01	1
Garlic	2	2021-05-01	1
Garlic	2	2021-06-01	1
Garlic	2	2021-07-01	1
Garlic	2	2021-08-01	1
Garlic	2	2021-08-09	1
Garlic	2	2021-08-14	1
Garlic	2	2021-09-01	1
Garlic	2	2021-09-07	1
Garlic	2	2021-09-23	1
Garlic	2	2021-10-01	1
Garlic	3	2021-06-01	-20
Garlic	3	2021-07-01	-20
Garlic	3	2021-08-01	-20
Garlic	3	2021-08-09	-20
Garlic	3	2021-08-14	-20
Garlic	3	2021-09-01	-20
Garlic	3	2021-09-07	-20
Garlic	3	2021-09-23	-20
Garlic	3	2021-10-01	-20
Grape	1	2021-01-01	99
Grape	1	2021-02-01	99
Grape	1	2021-03-01	99
Grape	1	2021-04-01	99
Grape	1	2021-05-01	99
Grape	1	2021-06-01	99
Grape	1	2021-07-01	104
Grape	1	2021-08-01	104
Grape	1	2021-08-09	104
Grape	1	2021-08-14	104
Grape	1	2021-09-01	104
Grape	1	2021-09-07	24
Grape	3	2021-01-01	10
Grape	3	2021-02-01	10
Grape	3	2021-03-01	10
Grape	3	2021-04-01	10
Grape	3	2021-05-01	10
Grape	3	2021-06-01	10
Grape	3	2021-07-01	5
Grape	3	2021-08-01	5

Grape	3	2021-08-09	5
Grape	3	2021-08-14	5
Grape	3	2021-09-01	5
Lemonade	1	2021-01-01	6
Lemonade	1	2021-02-01	12
Lemonade	1	2021-03-01	18
Lemonade	1	2021-04-01	24
Lemonade	1	2021-05-01	30
Lemonade	1	2021-06-01	16
Lemonade	1	2021-07-01	22
Lemonade	1	2021-08-01	28
Lemonade	1	2021-08-09	28
Lemonade	1	2021-08-14	28
Lemonade	1	2021-09-01	34
Lemonade	1	2021-09-07	34
Lemonade	1	2021-09-23	17
Lemonade	1	2021-10-01	23
Lemonade	3	2021-01-01	5
Lemonade	3	2021-02-01	5
Lemonade	3	2021-03-01	5
Lemonade	3	2021-04-01	5
Lemonade	3	2021-05-01	5
Lemonade	3	2021-06-01	25
Lemonade	3	2021-07-01	25
Lemonade	3	2021-08-01	25
Lemonade	3	2021-08-09	25
Lemonade	3	2021-08-14	25
Lemonade	3	2021-09-01	25
Lemonade	3	2021-09-07	8
Lemonade	3	2021-09-23	8
Lemonade	3	2021-10-01	8
Rice	1	2021-01-01	2
Rice	1	2021-02-01	2
Rice	1	2021-03-01	2
Rice	1	2021-04-01	2
Rice	1	2021-05-01	2
Rice	1	2021-06-01	2
Rice	1	2021-07-01	2
Rice	1	2021-08-01	2
Rice	1	2021-08-09	1
Rice	1	2021-08-14	1
Rice	1	2021-09-01	1
Rice	2	2021-01-01	2
Rice	2	2021-02-01	2

Rice	2	2021-03-01	2
Rice	2	2021-04-01	2
Rice	2	2021-05-01	2
Rice	2	2021-06-01	2
Rice	2	2021-07-01	2
Rice	2	2021-08-01	2
Rice	2	2021-08-09	2
Rice	2	2021-08-14	1
Rice	2	2021-09-01	1
Rice	2	2021-09-07	1
Rice	2	2021-09-23	1
Rice	2	2021-10-01	1
Rice	3	2021-01-01	8
Rice	3	2021-02-01	8
Rice	3	2021-03-01	8
Rice	3	2021-04-01	8
Rice	3	2021-05-01	8
Rice	3	2021-06-01	8
Rice	3	2021-07-01	8
Rice	3	2021-08-01	8
Rice	3	2021-08-09	8
Rice	3	2021-08-14	8
Rice	3	2021-09-01	8
Rice	3	2021-09-07	8
Rice	3	2021-09-23	8
Rice	3	2021-10-01	8

Test data

```
-- Data about customers

INSERT INTO Customers VALUES('1983-02-11','Jamie','Johnson',1);

INSERT INTO Customers VALUES('1995-10-26','Birgit','Doe',2);

INSERT INTO Customers VALUES('1991-05-15','Finn','Smith',3);

INSERT INTO Customers VALUES('1990-07-03','Anita','Taylor',4);


-- Data about employees

INSERT INTO Employees VALUES('1964-12-01','Carla','Brown',1);

INSERT INTO Employees VALUES('1984-03-14','Bryan','Williams',2);

INSERT INTO Employees VALUES('1991-02-19','Finn','Wilson',3);

INSERT INTO Employees VALUES('1998-08-12','Denise','Davies',4);
```

```sql
-- Data about locations
INSERT INTO Locations VALUES('Park Road 7',1);
INSERT INTO Locations VALUES('Hill Street 2',2);
INSERT INTO Locations VALUES('Duckinfield Street 5',3);

-- Data about transactions
INSERT INTO Transactions VALUES(1,3,1,'2021-08-09',1);
INSERT INTO Transactions VALUES(4,2,2,'2021-08-14',2);
INSERT INTO Transactions VALUES(4,4,1,'2021-09-07',3);
INSERT INTO Transactions VALUES(3,4,1,'2021-09-07',4);
INSERT INTO Transactions VALUES(4,1,3,'2021-09-07',5);
INSERT INTO Transactions VALUES(1,4,1,'2021-09-23',6);

-- Data about items
INSERT INTO Items VALUES(200,'Bread');
INSERT INTO Items VALUES(100,'Lemonade');
INSERT INTO Items VALUES(100,'Banana');
INSERT INTO Items VALUES(200,'Rice');
INSERT INTO Items VALUES(150,'Grape');
INSERT INTO Items VALUES(450,'Chicken');
INSERT INTO Items VALUES(25,'Garlic');

-- Data about items brought into shop
INSERT INTO ItemsBroughtIntoShop VALUES('Bread',1,13,'2021-1-1');
INSERT INTO ItemsBroughtIntoShop VALUES('Lemonade',1,6,'2021-1-1');
INSERT INTO ItemsBroughtIntoShop VALUES('Lemonade',1,6,'2021-2-1');
INSERT INTO ItemsBroughtIntoShop VALUES('Lemonade',1,6,'2021-3-1');
INSERT INTO ItemsBroughtIntoShop VALUES('Lemonade',1,6,'2021-4-1');
INSERT INTO ItemsBroughtIntoShop VALUES('Lemonade',1,6,'2021-5-1');
INSERT INTO ItemsBroughtIntoShop VALUES('Lemonade',1,6,'2021-6-1');
INSERT INTO ItemsBroughtIntoShop VALUES('Lemonade',1,6,'2021-7-1');
INSERT INTO ItemsBroughtIntoShop VALUES('Lemonade',1,6,'2021-8-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Lemonade',1,6,'2021-9-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Lemonade',1,6,'2021-10-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Banana',1,5,'2021-1-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Rice',1,2,'2021-1-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Grape',1,99,'2021-1-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Chicken',1,2,'2021-1-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Garlic',1,1,'2021-1-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Bread',1,3,'2021-1-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Banana',2,7,'2021-1-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Rice',2,2,'2021-1-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Chicken',2,19,'2021-1-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Garlic',2,1,'2021-1-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Bread',3,5,'2021-1-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Lemonade',3,5,'2021-1-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Banana',3,6,'2021-1-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Rice',3,8,'2021-1-1');

INSERT INTO ItemsBroughtIntoShop VALUES('Grape',3,10,'2021-1-1');


-- Data about movement of items

INSERT INTO MovementOfItems VALUES('Lemonade',1,3,20,'2021-6-1');

INSERT INTO MovementOfItems VALUES('Garlic',3,1,20,'2021-6-1');

INSERT INTO MovementOfItems VALUES('Banana',2,3,5,'2021-6-1');

INSERT INTO MovementOfItems VALUES('Banana',3,1,9,'2021-7-1');

INSERT INTO MovementOfItems VALUES('Banana',1,2,4,'2021-9-1');

INSERT INTO MovementOfItems VALUES('Grape',3,1,5,'2021-7-1');


-- Data about items in transactions

INSERT INTO ItemsInTransactions VALUES('Garlic',1,5);

INSERT INTO ItemsInTransactions VALUES('Bread',1,8);

INSERT INTO ItemsInTransactions VALUES('Chicken',1,1);
INSERT INTO ItemsInTransactions VALUES('Rice',1,1);

INSERT INTO ItemsInTransactions VALUES('Banana',2,3);

INSERT INTO ItemsInTransactions VALUES('Chicken',2,5);

INSERT INTO ItemsInTransactions VALUES('Rice',2,1);

INSERT INTO ItemsInTransactions VALUES('Rice',3,1);

INSERT INTO ItemsInTransactions VALUES('Chicken',3,1);

INSERT INTO ItemsInTransactions VALUES('Garlic',4,7);

INSERT INTO ItemsInTransactions VALUES('Grape',4,80);

INSERT INTO ItemsInTransactions VALUES('Lemonade',5,17);

INSERT INTO ItemsInTransactions VALUES('Grape',5,5);

INSERT INTO ItemsInTransactions VALUES('Lemonade',6,17);

INSERT INTO ItemsInTransactions VALUES('Grape',6,24);
```