DATABASE

MANAGEMENT

SYSTEMS DESIGN

By : Saksham Jain

**Part 1:**

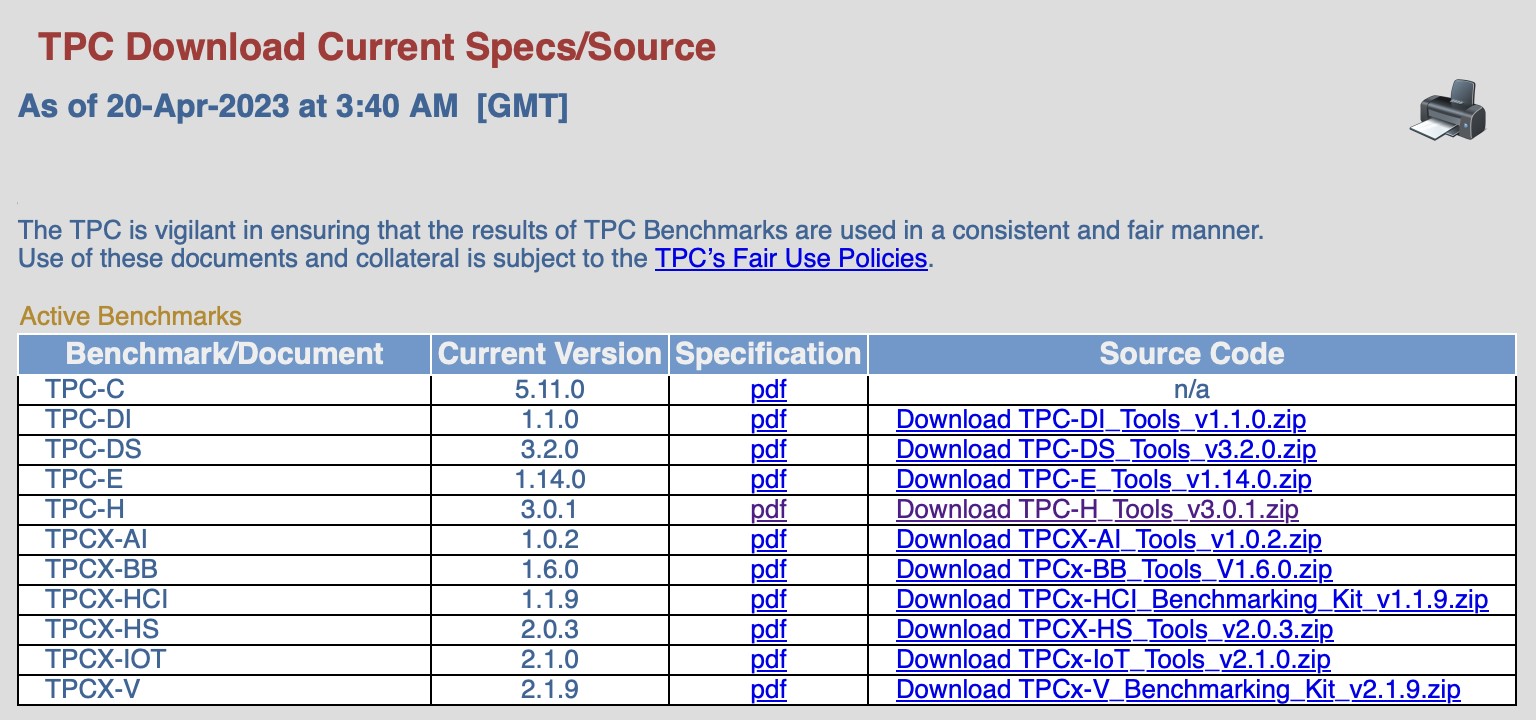
**Install TPC-H database (https://www.tpc.org/tpch/) in your local computer/laptop and use scale factor s= 0.001. Provide detailed instructions, as well as the reasons for using those instructions with screenshots of how you installed TPCH on your computer. (30 points)**

**You need to create the 8 tables given in TPCH. (1 point) Write down the primary key and foreign key of each table. (1 points)**

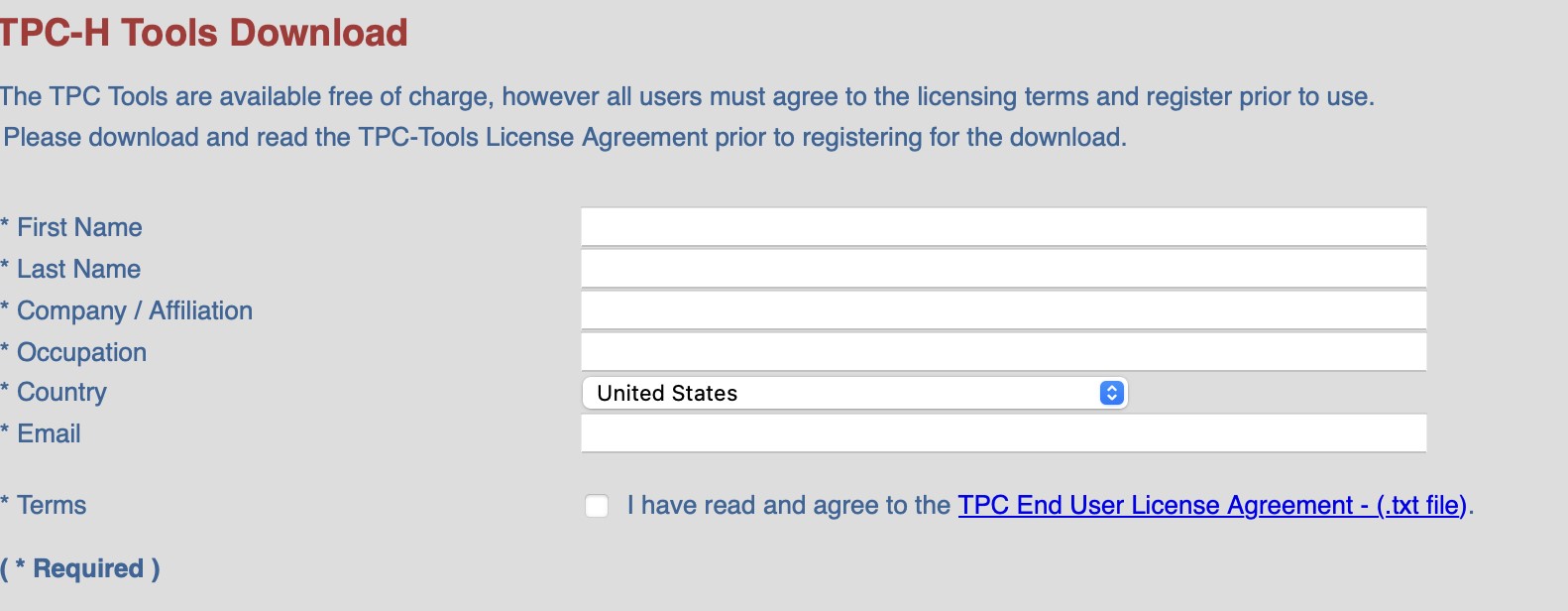
**Let’s First download the TPCH dataset**

**Steps to download TPC-H are as follows :**

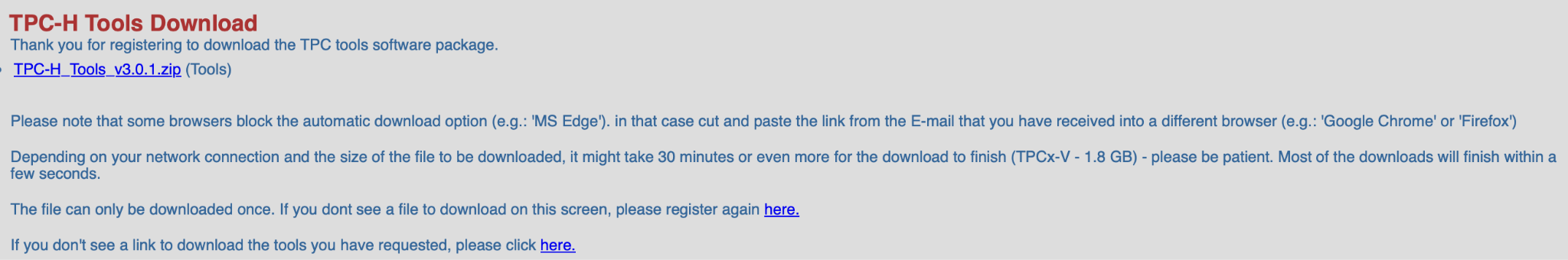
Step 1 : Copy the TPC-H link given in the assignment sheet. TPC website will open. Step 2 : Go to the Downloads tab in the website and select TPC-H Version 3.0.1. As shown in the image below.



Step 3 : As shown below, this tab will open. Kindly fill details. We will receive an email containing the link to download the tools. The link is an one time useable only and valid for a limited period of time. You will get a zip file from this link, download that zip file by clicking on it.



Step 4 : As shown below, this tab will open. Click on the link and the TPC-H zip file will be downloaded to your laptop.



Step 5 : Now open Command prompt (terminal for mac OS). In terminal window, browse to the folder having TPC-H zip file.

Step 6 : As shown below, using ‘cd’ command, we have pointed to dbgen file inside TPCH. Now see all the files inside zip file using ‘ls’ command.

Step 7 : Now we copy and rename makefile.suite

Command : **cp makefile.suite makefile**

I’m using MAC and for database I’m using MYSQL so below are the values I added in the file. **CC = gcc DATABASE =SQLSERVER**

**MACHINE = LINUX**

**WORKLOAD = TPCH**

After making the changes save it and on the terminal run the command “make”.

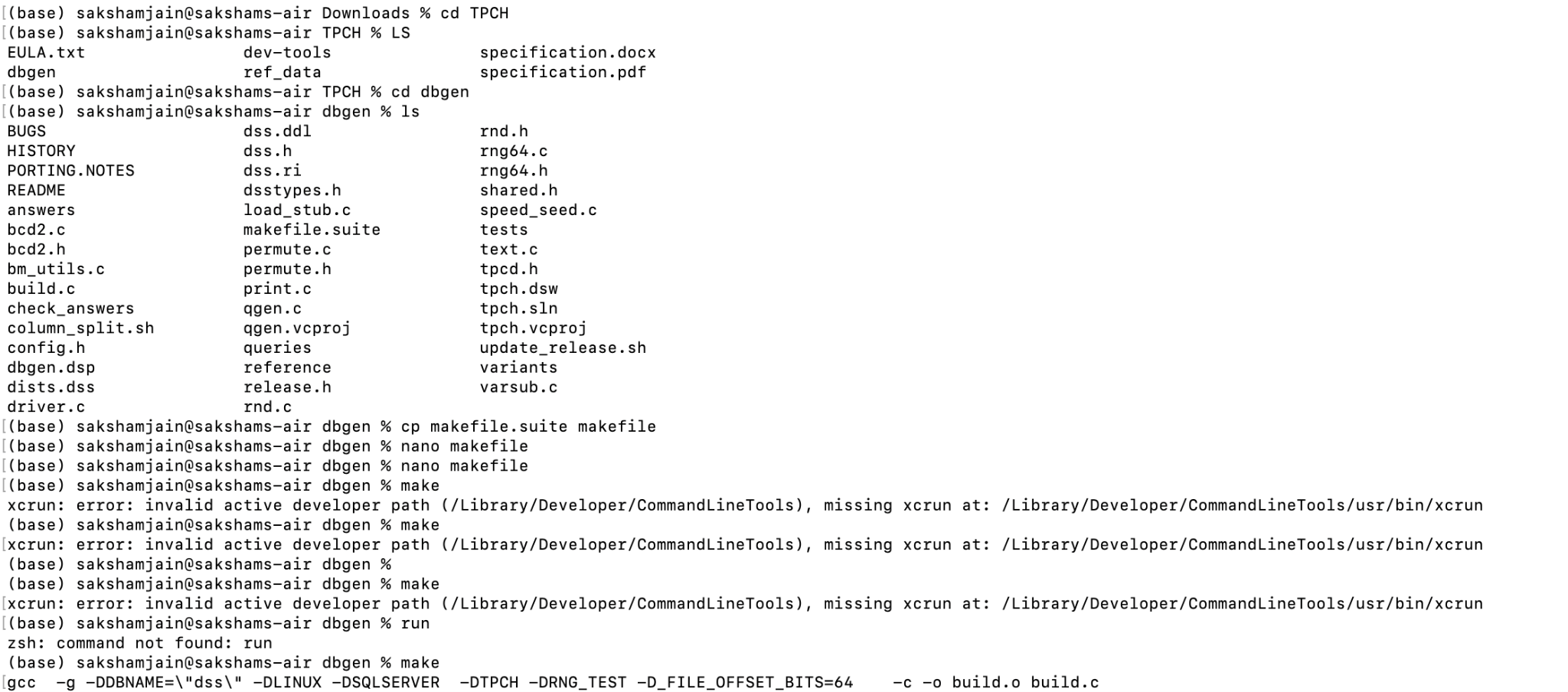


Step 8 : We have used command “**nano**”. Command nano is an easy to use command line text editor for Linux operating system. It includes all the basic functionality from a regular text editor, like syntax highlighting, multiple buffers, search and replace with regular expression support, UTF-8 encoding.

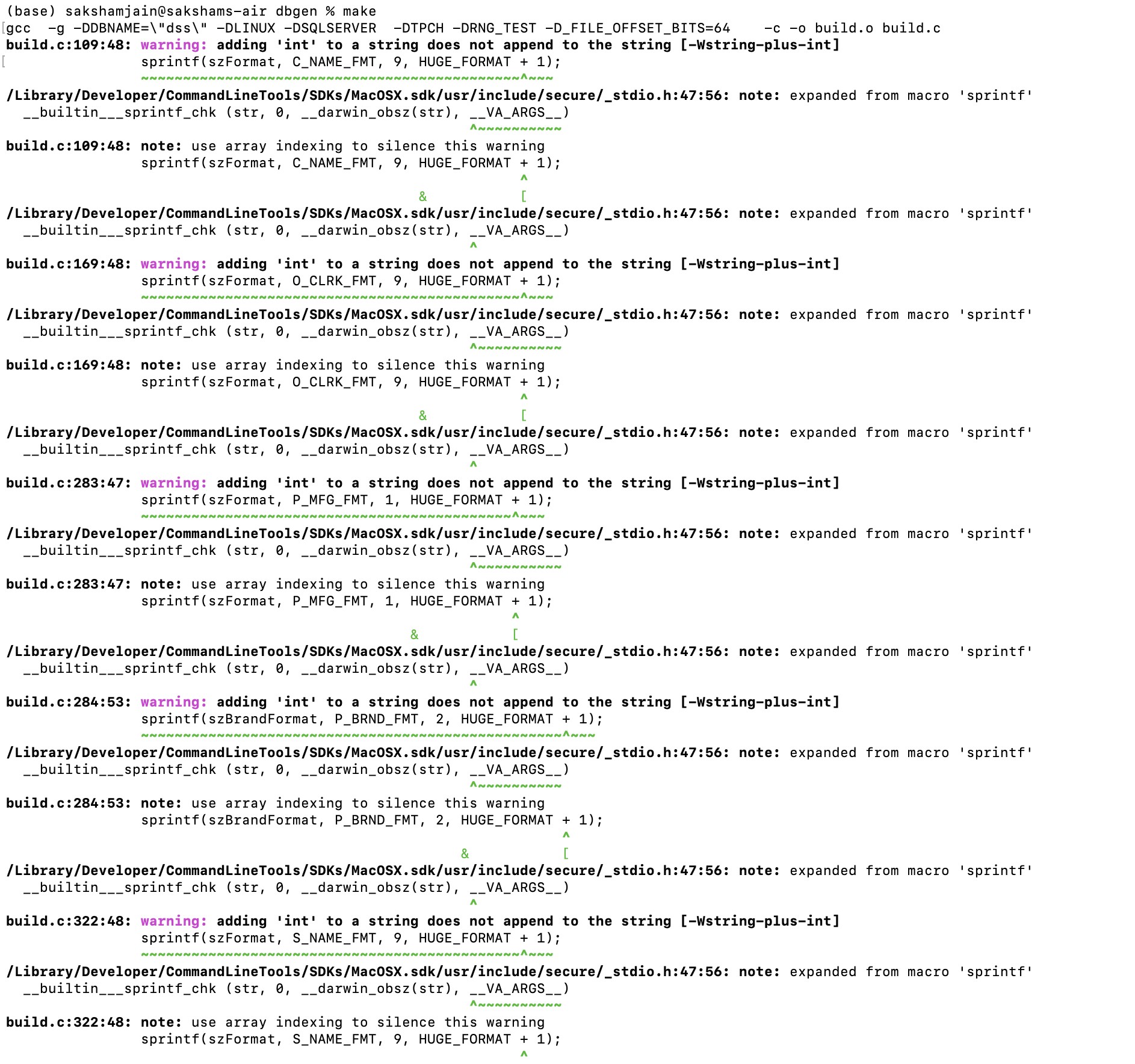
Step 9 : Run this using ‘**make**’ command. “**make**” command has created the dbgen utility successfully. Now we will be generating the data using the scale factor s=0.001, this will produce a small amount of test data and place all the newly created tables in the same directory I.e, in dbgen.

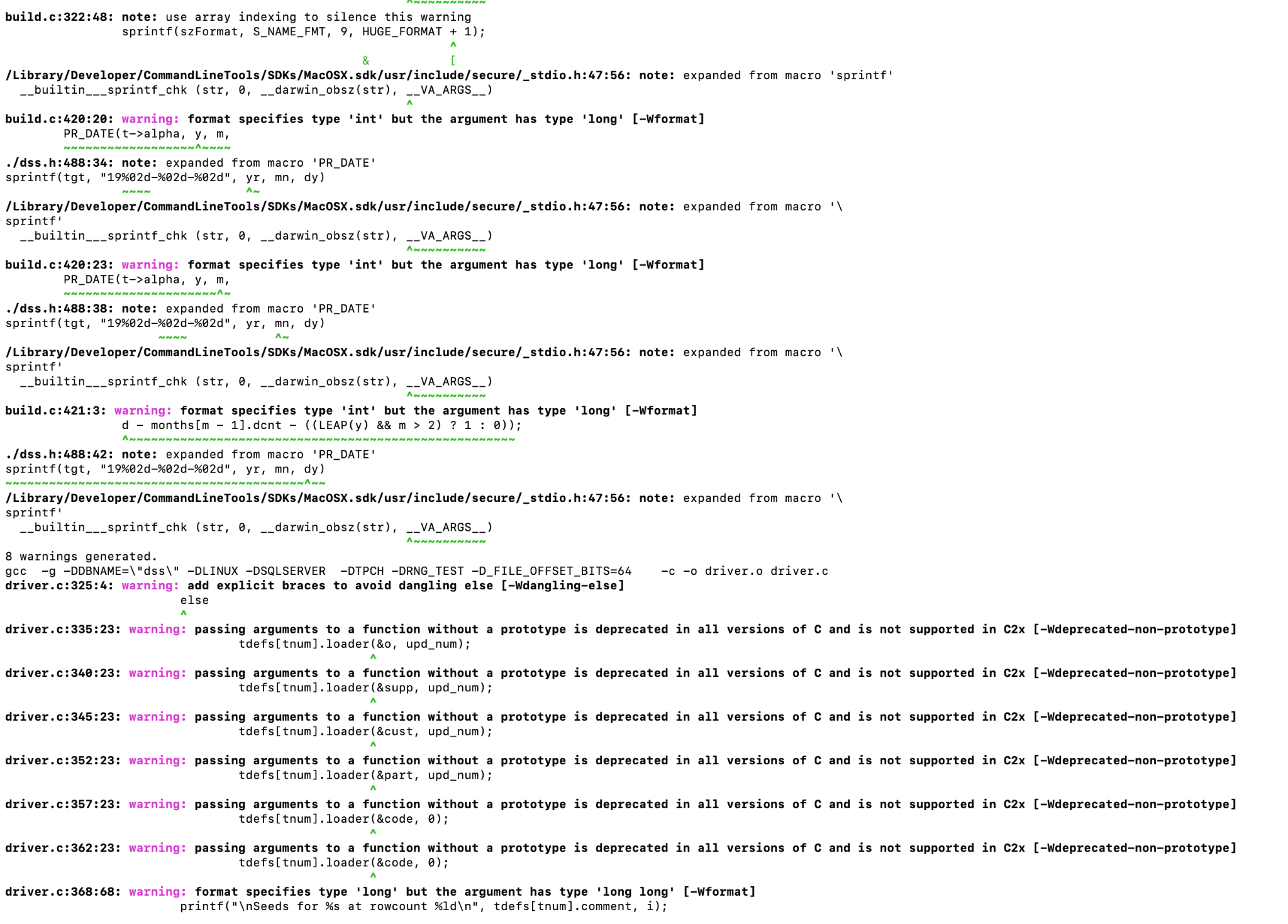
Command to produce the data is

**./dbgen -s 0.001**



Step 10 : As you can see, the command gets executed as shown below.

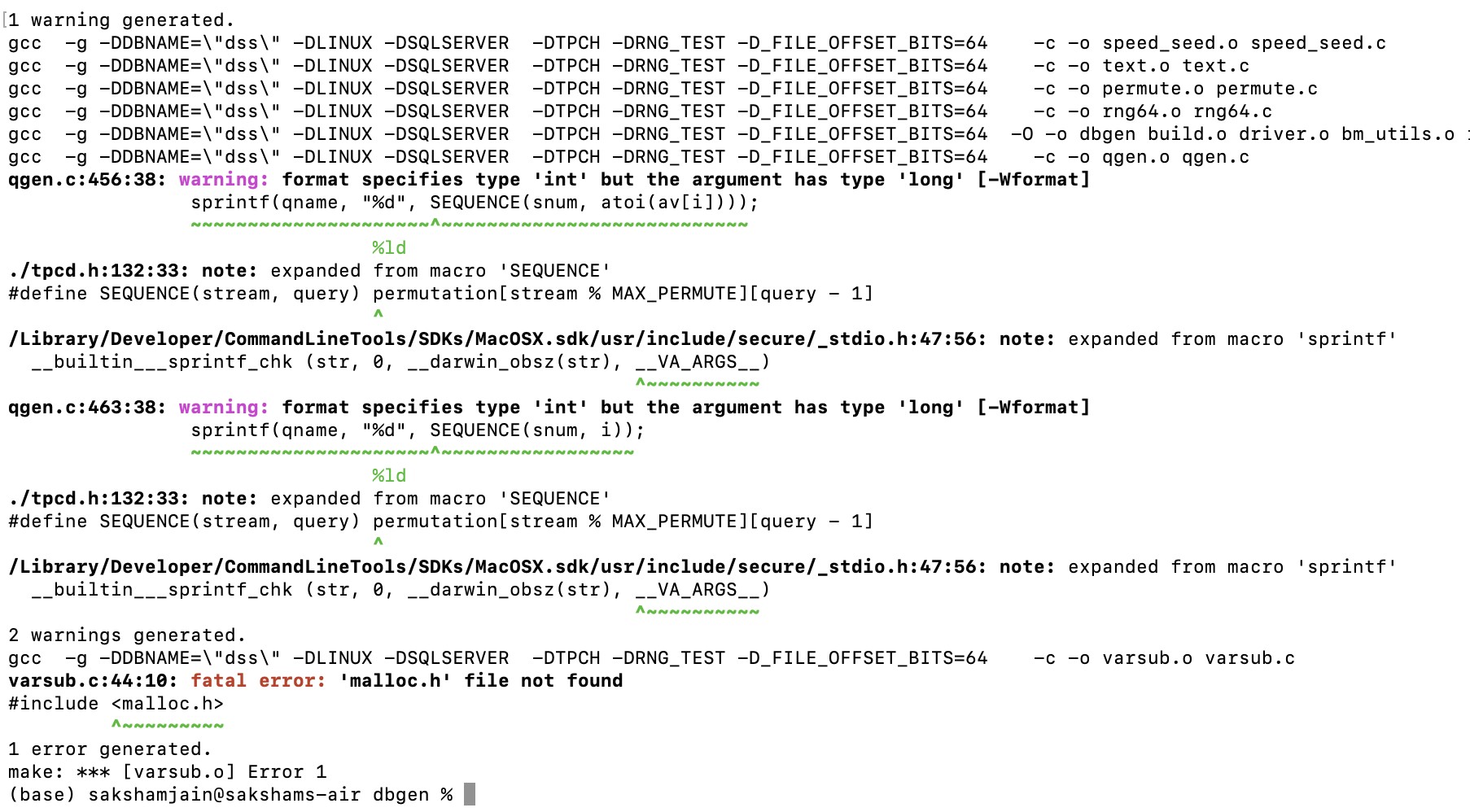




Step 11 : We get some **errors**, so we make changes in file ‘bm\_utils.c’ .

Step 12 : We get another error and hence we change machine.

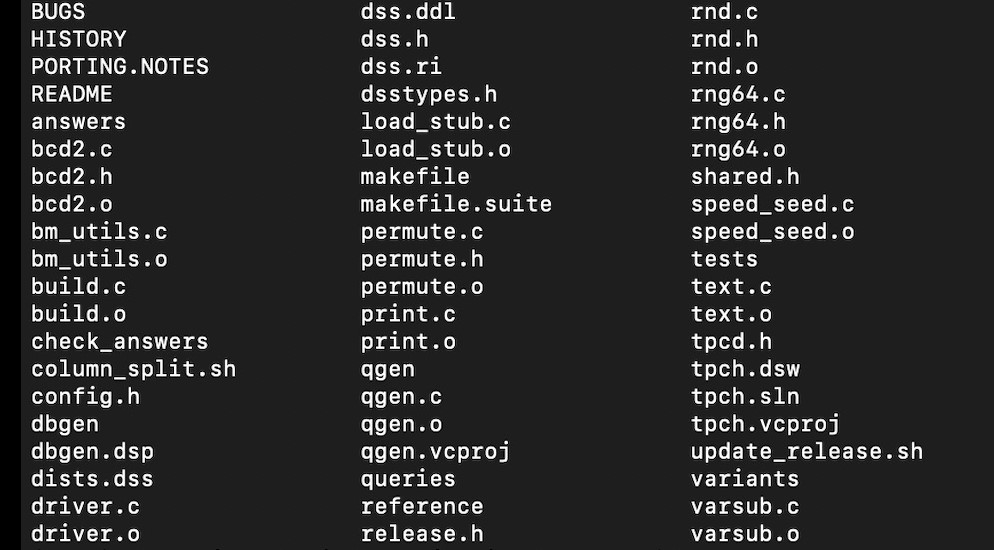
We take another mac OS laptop and run the above steps again. Following the steps from 1 - 11.



Step 13 : We have performed same steps in below new machine.



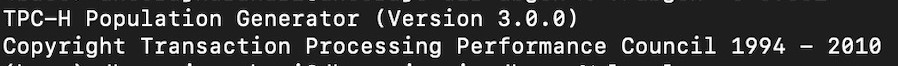
Step 13 : As you can see, now we are able to get the error resolved in 2nd machine. Now we see all files in TPC-H.



Step 14 : Now we use scale factor = 0.001. As you can see in below image, the TPCH population generator generates the files. Command to produce the data is

**./dbgen -s 0.001**





Step 15 : You can finally see the tables of TPC-H dataset. The tables are ending with “ .tbl” extension. In images below you can see 8 tables of dataset.

**Namely :**

1. **Supplier**
2. **Region**
3. **Partsupp**
4. **Part**
5. **Orders**
6. **Nation**
7. **Lineitem**
8. **Customers**

Now that we have got our tables. Let’s export the data to mysql server and create the 8 tables in mysql.

Step 1 : We got the required 8 tables, now convert these files to cvs format so that we can load those files directly in the MYSQL. We have used **https://dev.mysql.com/doc/heatwave/en/mys-hw-tpchquickstart-create-database-import.html** to get the sample tables.

Step 2 : By providing the .csv table file we can load the entire data. Use the following code

LOAD DATA INFILE

‘/Users/sakshamjain/Downloads/TPC\_H\_V3.0.1/dbgen/customer.csv’ INTO TABLE customer

FIELDS TERMINATED BY ‘|’ ENCLOSED BY ‘“’

LINES TERMINATED BY ‘\n’

Step 3 : Generate Output. As you can see the output of our create tables.



So, we have successfully installed TPC-H tools and loaded it into SQL. Now the data set is ready to use.

CREATE TABLE **nation** ( N\_NATIONKEY INTEGER primary key, N\_NAME

CHAR(25) NOT NULL, N\_REGIONKEY INTEGER NOT NULL, N\_COMMENT

VARCHAR(152));

CREATE TABLE **region** ( R\_REGIONKEY INTEGER primary key, R\_NAME

CHAR(25) NOT NULL, R\_COMMENT VARCHAR(152));

CREATE TABLE **part** ( P\_PARTKEY INTEGER primary key, P\_NAME

VARCHAR(55) NOT NULL, P\_MFGR CHAR(25) NOT NULL, P\_BRAND CHAR(10)

NOT NULL, P\_TYPE VARCHAR(25) NOT NULL, P\_SIZE INTEGER NOT NULL,

P\_CONTAINER CHAR(10) NOT NULL, P\_RETAILPRICE DECIMAL(15,2) NOT

NULL, P\_COMMENT VARCHAR(23) NOT NULL );

CREATE TABLE **supplier** ( S\_SUPPKEY INTEGER primary key, S\_NAME

CHAR(25) NOT NULL, S\_ADDRESS VARCHAR(40) NOT NULL, S\_NATIONKEY

INTEGER NOT NULL, S\_PHONE CHAR(15) NOT NULL, S\_ACCTBAL

DECIMAL(15,2) NOT NULL, S\_COMMENT VARCHAR(101) NOT NULL);

CREATE TABLE **partsupp** ( PS\_PARTKEY INTEGER NOT NULL, PS\_SUPPKEY

INTEGER NOT NULL, PS\_AVAILQTY INTEGER NOT NULL, PS\_SUPPLYCOST DECIMAL(15,2) NOT NULL, PS\_COMMENT VARCHAR(199) NOT NULL, primary key (ps\_partkey, ps\_suppkey) );

CREATE TABLE **customer** ( C\_CUSTKEY INTEGER primary key, C\_NAME

VARCHAR(25) NOT NULL, C\_ADDRESS VARCHAR(40) NOT NULL,

C\_NATIONKEY INTEGER NOT NULL, C\_PHONE CHAR(15) NOT NULL,

C\_ACCTBAL DECIMAL(15,2) NOT NULL, C\_MKTSEGMENT CHAR(10) NOT

NULL, C\_COMMENT VARCHAR(117) NOT NULL);

CREATE TABLE **orders** ( O\_ORDERKEY INTEGER primary key, O\_CUSTKEY

INTEGER NOT NULL, O\_ORDERSTATUS CHAR(1) NOT NULL, O\_TOTALPRICE

DECIMAL(15,2) NOT NULL, O\_ORDERDATE DATE NOT NULL,

O\_ORDERPRIORITY CHAR(15) NOT NULL, O\_CLERK CHAR(15) NOT NULL,

O\_SHIPPRIORITY INTEGER NOT NULL, O\_COMMENT VARCHAR(79) NOT NULL);

CREATE TABLE **lineitem** ( L\_ORDERKEY INTEGER NOT NULL, L\_PARTKEY

INTEGER NOT NULL, L\_SUPPKEY INTEGER NOT NULL, L\_LINENUMBER

INTEGER NOT NULL, L\_QUANTITY DECIMAL(15,2) NOT NULL,

L\_EXTENDEDPRICE DECIMAL(15,2) NOT NULL, L\_DISCOUNT DECIMAL(15,2)

NOT NULL, L\_TAX DECIMAL(15,2) NOT NULL, L\_RETURNFLAG CHAR(1) NOT

NULL, L\_LINESTATUS CHAR(1) NOT NULL, L\_SHIPDATE DATE NOT NULL,

L\_COMMITDATE DATE NOT NULL, L\_RECEIPTDATE DATE NOT NULL,

L\_SHIPINSTRUCT CHAR(25) NOT NULL, L\_SHIPMODE CHAR(10) NOT NULL, L\_COMMENT VARCHAR(44) NOT NULL, primary key(L\_ORDERKEY,L\_LINENUMBER));

|  |  |  |
| --- | --- | --- |
| Table Name | Primary Key | Foreign Key |
| nation | N\_NATIONKEY | N\_REGIONKEY |
| region | R\_REGIONKEY |  |
| part | P\_PARTKEY |  |

|  |  |  |
| --- | --- | --- |
| supplier | S\_SUPPKEY | S\_NATIONKEY |
| partsupp | PS\_PARTKEY, PS\_SUPPKEY |  |
| customer | C\_CUSTKEY | C\_NATIONKEY |
| orders | O\_ORDERKEY | O\_CUSTKEY |
| lineitem | L\_ORDERKEY,L\_LINENUMBER | L\_PARTKEY,L\_SUPPKEY |



**Part B:**

**A special query: count the number of rows in each table of TPCH. You must provide the command and screenshot of the output of the command in the answer sheet. If you are not able to generate data at scale\_factor =0.001, then the answer to this query will be wrong. And if the answer to this query is wrong, then the answers to the following query will also be wrong, since you have not generated the data correctly. You do not need to write a relational algebra expression for this query. B2. How many “part\_key” are produced by “nationkey=5” and “7” or n\_name= India for each P\_brand?**

**B3. How many items are delivered “DELIVER IN PERSON” or “TAKE BACK RETURN” between “1994-01-23” and “1996-02-1?”**

**Question 5 Perform the following queries using SQL:**

**5.1 Create view account\_loan which has 3 columns: account\_id, number of loans, total sum of loans. 682 rows)**

CREATE VIEW account\_loan AS

SELECT

l.account\_id,

COUNT(l.id) AS num\_loans,

SUM(l.amount) AS total\_sum\_loans

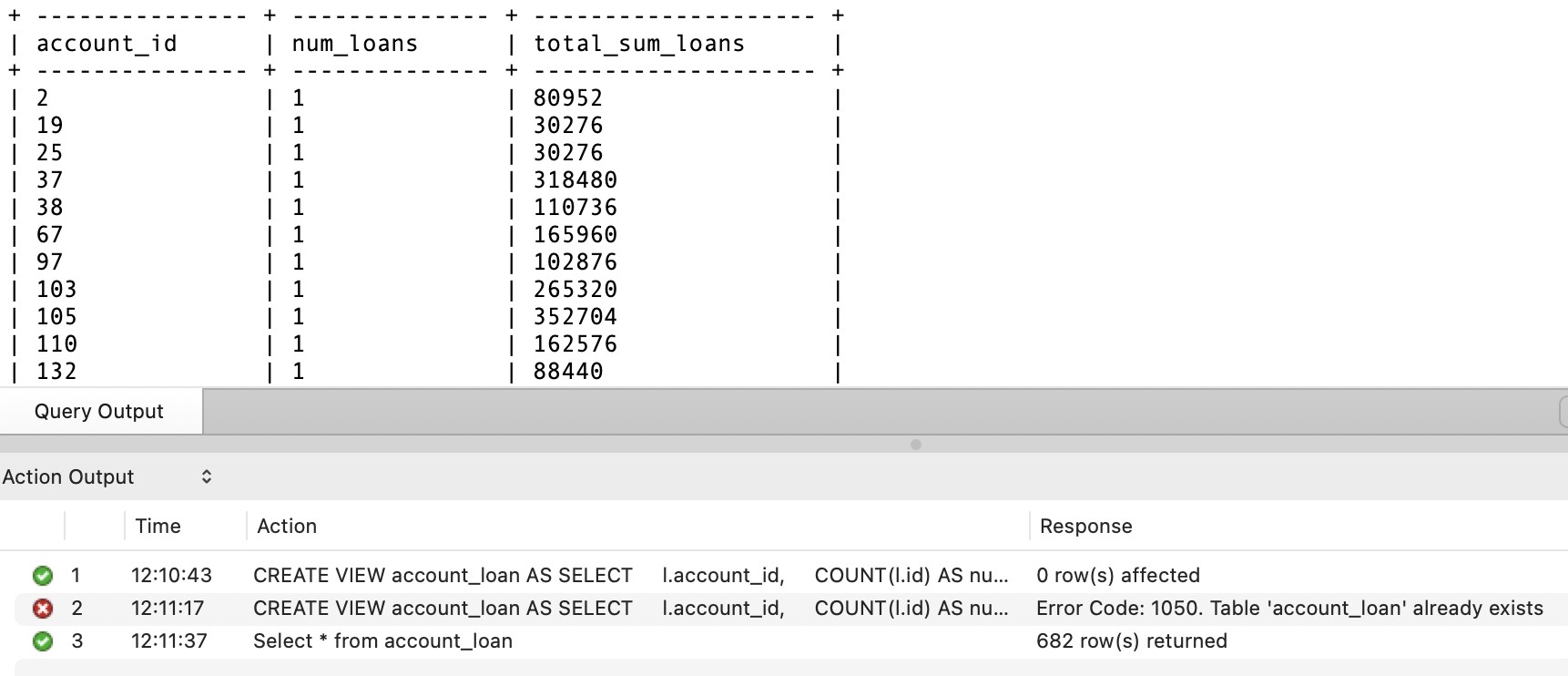
FROM

Loans l

GROUP BY

l.account\_id;

Select \* from account\_loan;



**5.2 Create view account\_order which has 3 columns: account\_id, number of orders, total sum orders. (3758 rows)**

CREATE VIEW account\_order AS

SELECT

o.account\_id,

COUNT(o.id) AS num\_orders,

**o**

SUM(o.amount) AS total\_sum\_orders

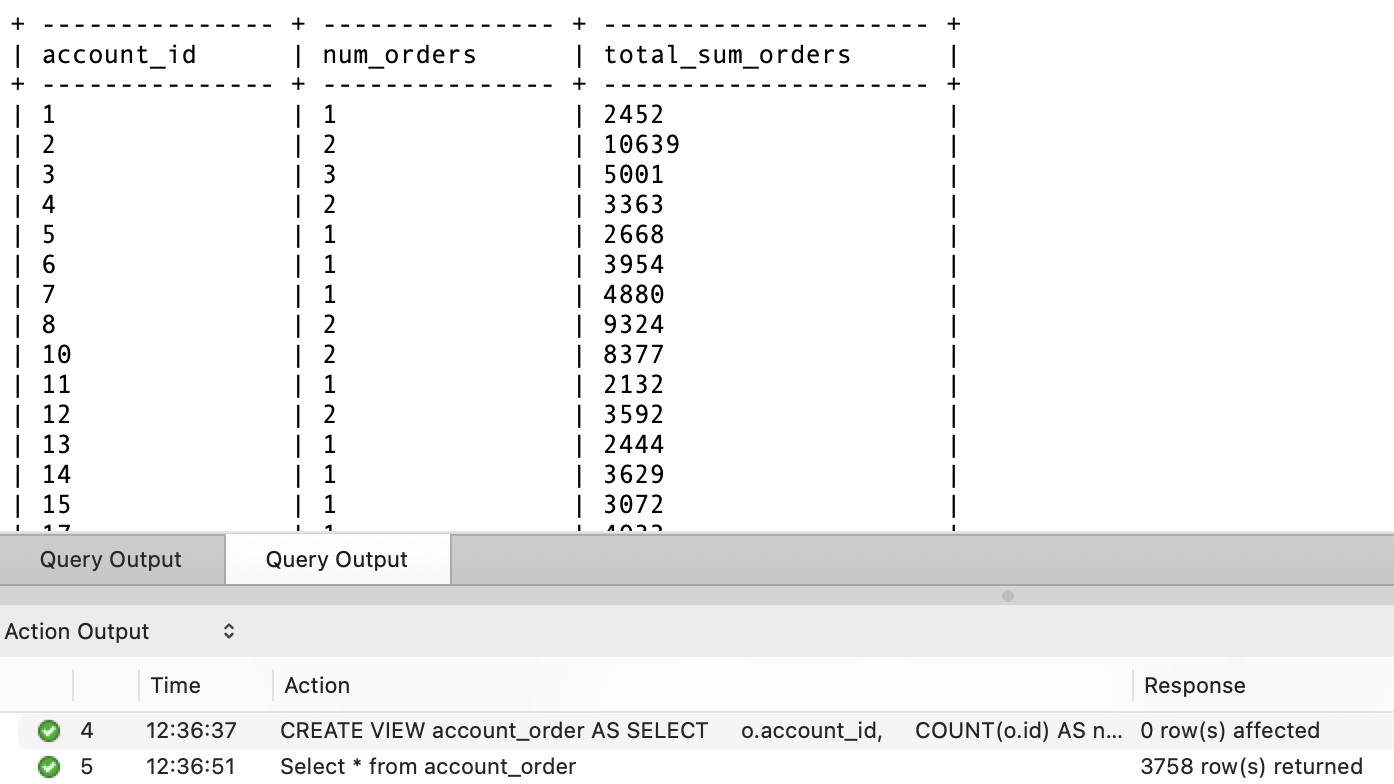
FROM

Orders o

GROUP BY

o.account\_id;

Select \* from account\_order;



**5.3 Create view account\_trans which has 3 columns: account\_id, number of transactions, total sum of transactions. (4500 rows)**

CREATE VIEW account\_trans AS

SELECT

t.account\_id,

COUNT(t.id) AS num\_transactions,

SUM(t.amount) AS total\_sum\_transactions

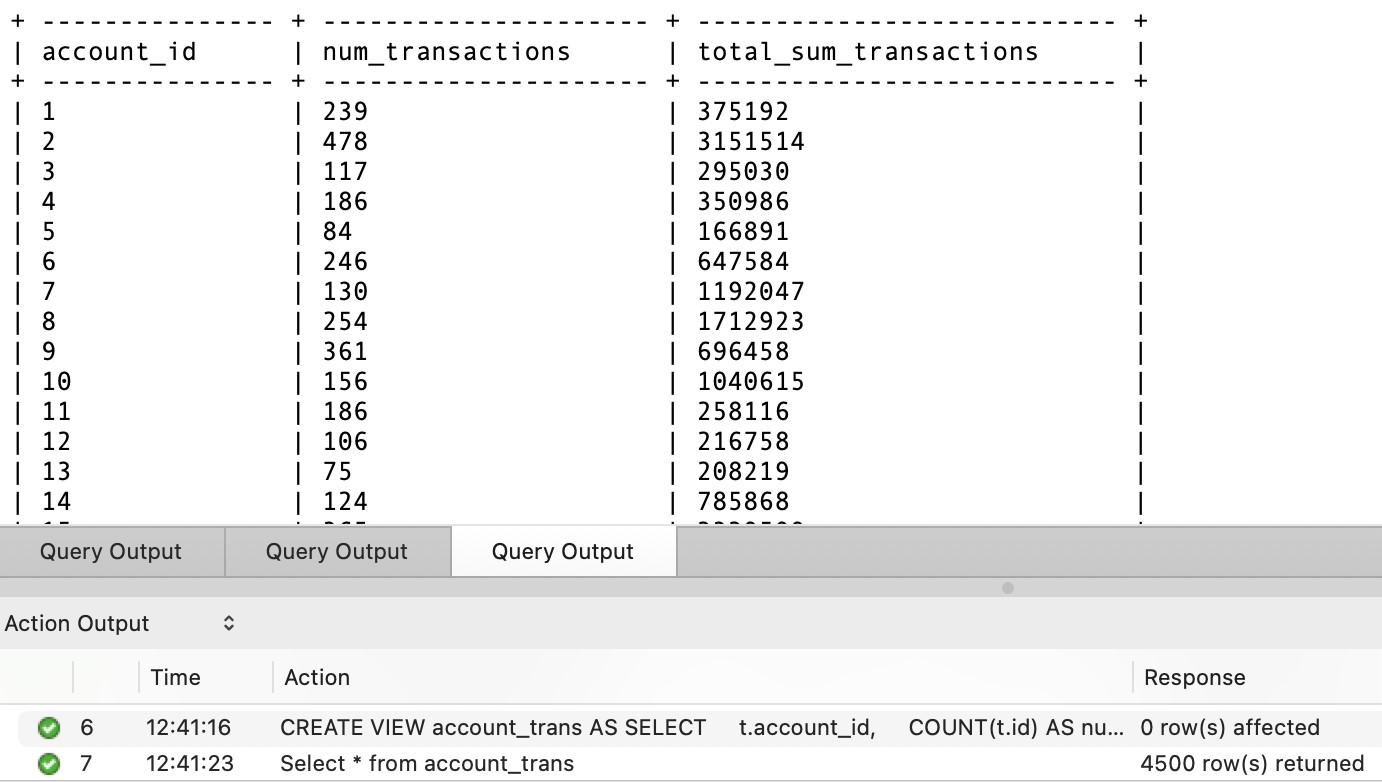
FROM

Transactions t

GROUP BY

t.account\_id;

Select \* from account\_trans;



**5.4 Create view account\_loan\_order\_trans which provides the number of loans, order, and transactions for each account along with the total amounts of each. (7576 rows) (Hint: use previous 3 views)**

CREATE VIEW account\_loan\_order\_trans AS

SELECT

a.account\_id,

COALESCE(al.num\_loans, 0) AS num\_loans,

COALESCE(al.total\_sum\_loans, 0) AS total\_sum\_loans,

COALESCE(ao.num\_orders, 0) AS num\_orders,

COALESCE(ao.total\_sum\_orders, 0) AS total\_sum\_orders, COALESCE(at.num\_transactions, 0) AS num\_transactions,

COALESCE(at.total\_sum\_transactions, 0) AS total\_sum\_transactions

FROM

Accounts a

LEFT JOIN

account\_loan al ON a.id = al.account\_id

LEFT JOIN

account\_order ao ON a.id = ao.account\_id

LEFT JOIN

account\_trans at ON a.id = at.account\_id;

Select \* from account\_loan\_order\_trans;

**5.5 Create a view district\_stats where each row stores district id, district name, stat\_type ( account or clients), count of the stat in district (# of accounts or # of clients), percentage of stat (# of accounts/# of clients / district population). (154 rows) (Percentage should be in percent form not decimal i.e., multiply by 100)**

CREATE VIEW district\_stats AS

SELECT

d.id AS district\_id,

d.name AS district\_name,

'account' AS stat\_type,

COALESCE(COUNT(DISTINCT a.id), 0) AS count\_stat,

COALESCE((COUNT(DISTINCT a.id) / NULLIF(COUNT(DISTINCT c.id), 0)) \* 100, 0) AS percentage\_stat

FROM

Accounts a

JOIN

Districts d ON a.district\_id = d.id

JOIN

Clients c ON a.district\_id = c.district\_id

GROUP BY

d.id, d.name

UNION ALL

SELECT

d.id AS district\_id,

d.name AS district\_name,

'client' AS stat\_type,

COALESCE(COUNT(DISTINCT c.id), 0) AS count\_stat,

COALESCE((COUNT(DISTINCT c.id) / NULLIF(COUNT(DISTINCT d.population), 0)) \* 100, 0) AS percentage\_stat

FROM

Clients c

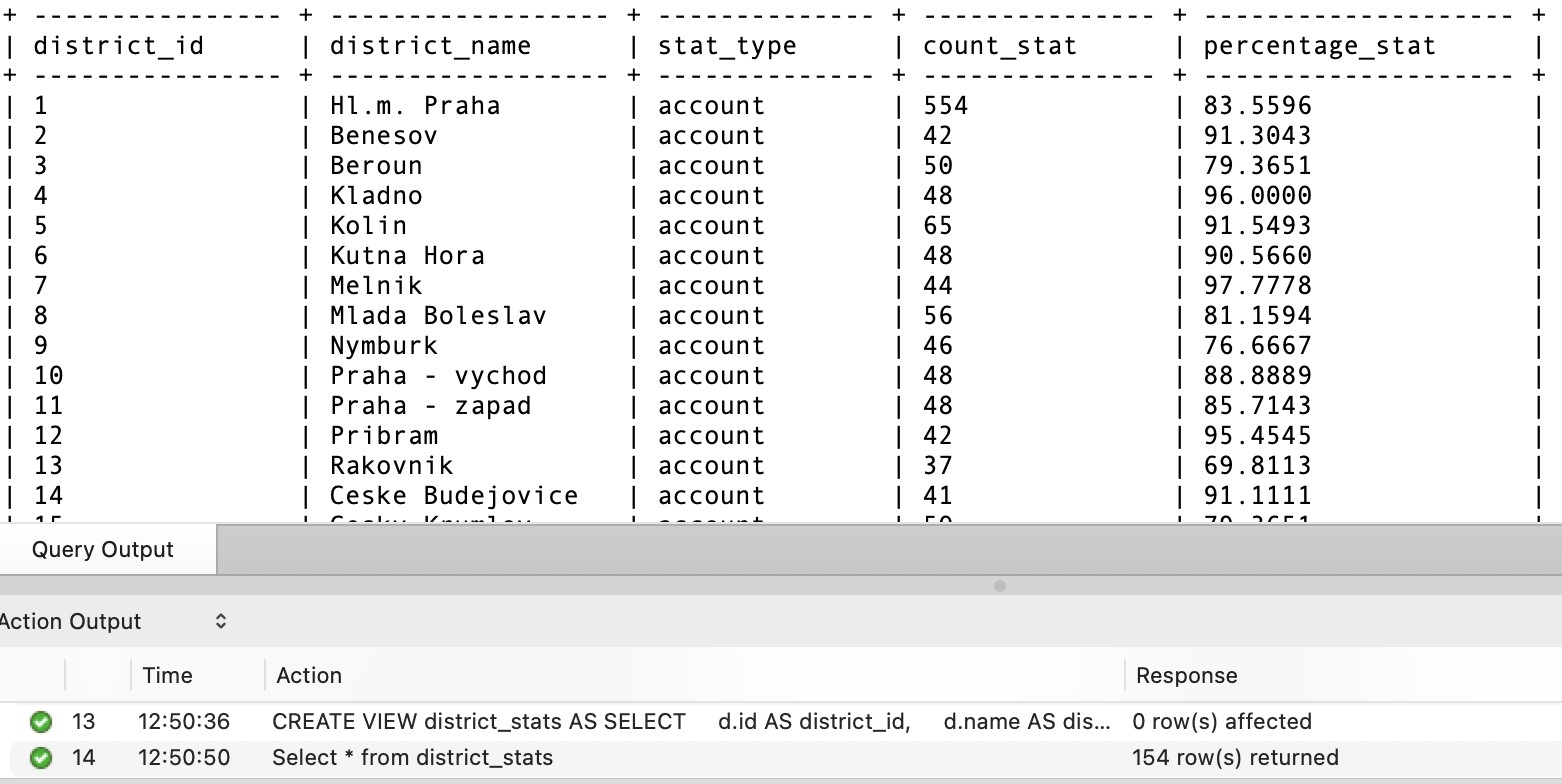
JOIN

Districts d ON c.district\_id = d.id

GROUP BY

d.id, d.name;

Select \* from district\_stats;



**5.6 Create view account\_stats with 5 columns: account\_id, type (loan/order/transaction), count of type, total amount of type, avg amount of type. One column for each account type pair. (8940 rows)**

CREATE VIEW account\_stats AS

SELECT

a.id AS account\_id,

'loan' AS type,

COALESCE(COUNT(l.id), 0) AS count\_type,

COALESCE(SUM(l.amount), 0) AS total\_amount\_type,

COALESCE(AVG(l.amount), 0) AS avg\_amount\_type

FROM

Loans l

JOIN

Accounts a ON l.account\_id = a.id

GROUP BY

a.id

UNION ALL

SELECT

a.id AS account\_id,

'order' AS type,

COALESCE(COUNT(o.id), 0) AS count\_type,

COALESCE(SUM(o.amount), 0) AS total\_amount\_type,

COALESCE(AVG(o.amount), 0) AS avg\_amount\_type

FROM

Orders o

JOIN

Accounts a ON o.account\_id = a.id

GROUP BY

a.id

UNION ALL

SELECT

a.id AS account\_id, 'transaction' AS type,

COALESCE(COUNT(t.id), 0) AS count\_type,

COALESCE(SUM(t.amount), 0) AS total\_amount\_type,

COALESCE(AVG(t.amount), 0) AS avg\_amount\_type

FROM

Transactions t

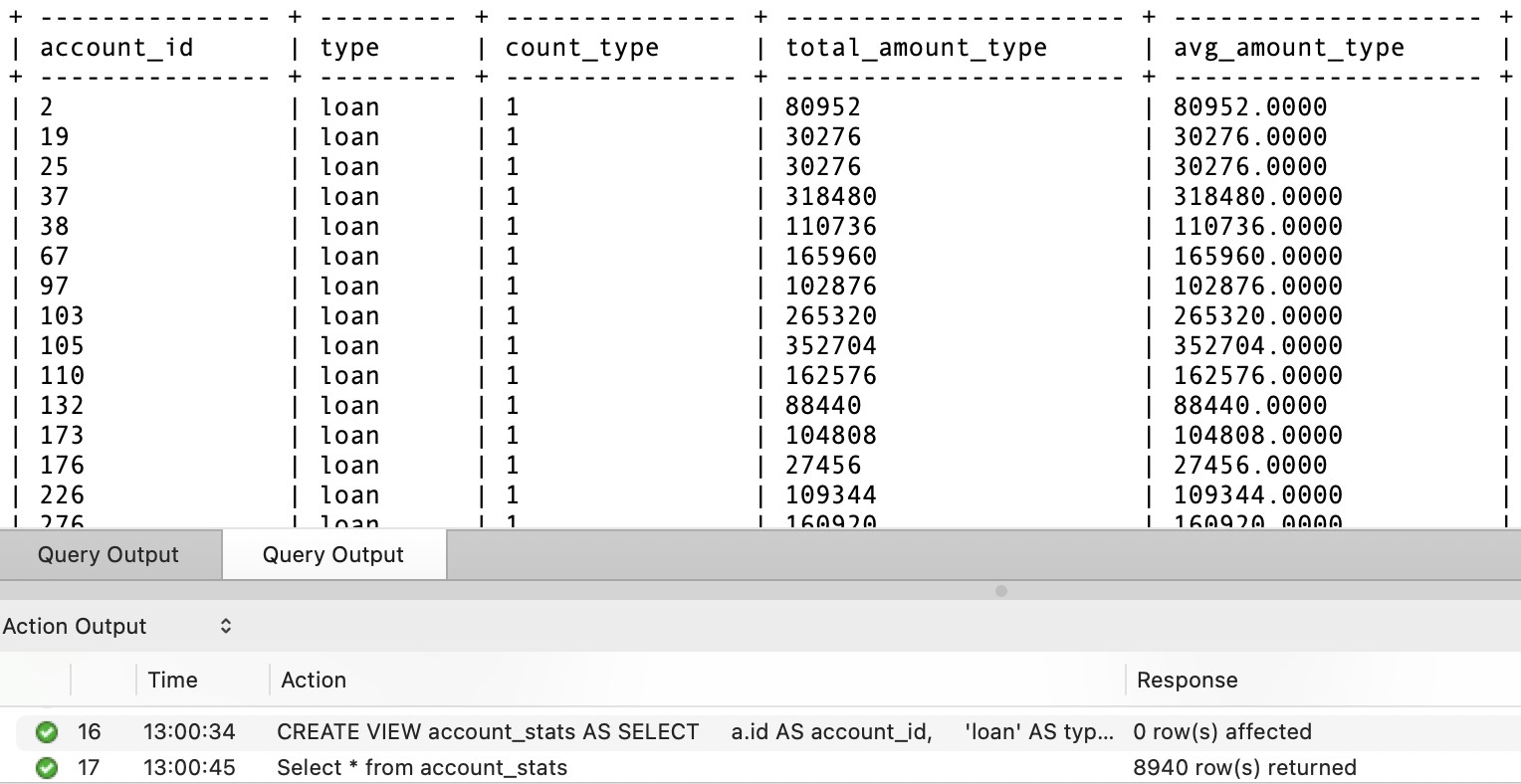
JOIN

Accounts a ON t.account\_id = a.id

GROUP BY

a.id;

Select \* from account\_stats;



**5.7 ) Create a trigger on transactions that executes before insert and stores the transaction in a table called flagged\_transactions (columns same as transactions) if the transaction amount is more than 2 times the average transaction amount for the account. Then insert the following rows and display all rows of flagged\_transactions.**

-- Create the flagged\_transactions table if it doesn't exist

CREATE TABLE IF NOT EXISTS flagged\_transactions ( id INT PRIMARY KEY AUTO\_INCREMENT,

account\_id INT, date DATE, type VARCHAR(255), operation VARCHAR(255), amount DECIMAL(10, 2), balance DECIMAL(10, 2), k\_symbol VARCHAR(255), bank VARCHAR(255), account VARCHAR(255)

);

-- Create the trigger on Transactions table

DELIMITER //

CREATE TRIGGER trg\_transactions\_before\_insert

BEFORE INSERT ON transactions FOR EACH ROW

BEGIN

DECLARE avg\_amount DECIMAL(10, 2);

-- Calculate the average transaction amount for the account

SELECT AVG(amount) INTO avg\_amount

FROM transactions

WHERE account\_id = NEW.account\_id;

-- Check if the transaction amount is more than 2 times the average IF NEW.amount > 2 \* avg\_amount THEN

-- Insert the flagged transaction into flagged\_transactions table

INSERT INTO flagged\_transactions (account\_id, date, type, operation, amount, balance, k\_symbol, bank, account)

VALUES (NEW.account\_id, NEW.date, NEW.type, NEW.operation, NEW.amount,

NEW.balance, NEW.k\_symbol, NEW.bank, NEW.account); END IF;

END;

//

DELIMITER ;

INSERT INTO TRANSACTIONS VALUES(4000005, 1, '1998-12-31', "VYDAJ", "VYBER", 1000,20000 "UROK", null, null);

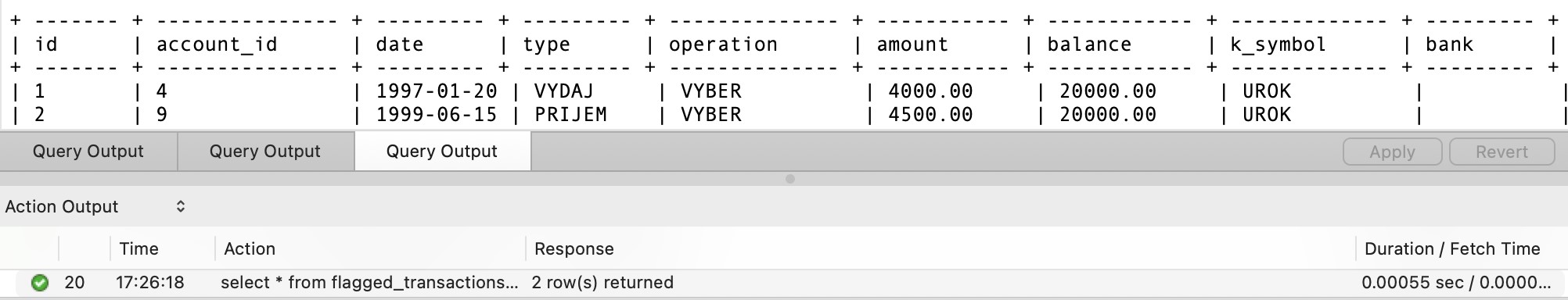
INSERT INTO TRANSACTIONS VALUES(4000006, 2, '1995-12-31', "VYDAJ", "VYBER", 3000,20000 "UROK", null, null);

INSERT INTO TRANSACTIONS VALUES(4000007, 4, '1997-1-20', "VYDAJ", "VYBER", 4000, 20000, "UROK", null, null);

INSERT INTO TRANSACTIONS VALUES(4000008, 6, '1998-5-12', "PRIJEM", "VYBER", 3000,20000, "UROK", null, null);

INSERT INTO TRANSACTIONS VALUES(4000009, 8, '1994-9-22', "PRIJEM", "VYBER", 500, 20000, "UROK", null, null);

INSERT INTO TRANSACTIONS VALUES(4000010, 9, '1999-6- 15', "PRIJEM", "VYBER", 4500, 20000 "UROK", null, null);



**5.8 ) Create a table called credit\_in\_cash with 3 columns: id (stores transaction id), account\_id, amount. Create a trigger called cash\_credit\_log that logs the id, account id, and amount of all cash credit transactions (operation = VKLAD). Run the following commands and then display credit\_in\_cash. (5 rows)**

-- Create the credit\_in\_cash table if it doesn't exist

CREATE TABLE IF NOT EXISTS credit\_in\_cash ( id INT PRIMARY KEY AUTO\_INCREMENT,

transaction\_id INT, account\_id INT,

amount DECIMAL(10, 2)

);

-- Create the trigger on Transactions table

DELIMITER //

CREATE TRIGGER trg\_cash\_credit\_log

AFTER INSERT ON transactions FOR EACH ROW

BEGIN

-- Check if the transaction is a cash credit transaction (operation = 'VKLAD') IF NEW.operation = 'VKLAD' THEN

-- Insert the transaction id, account id, and amount into credit\_in\_cash table

INSERT INTO credit\_in\_cash (transaction\_id, account\_id, amount)

VALUES (NEW.id, NEW.account\_id, NEW.amount); END IF;

END;

//

DELIMITER ;

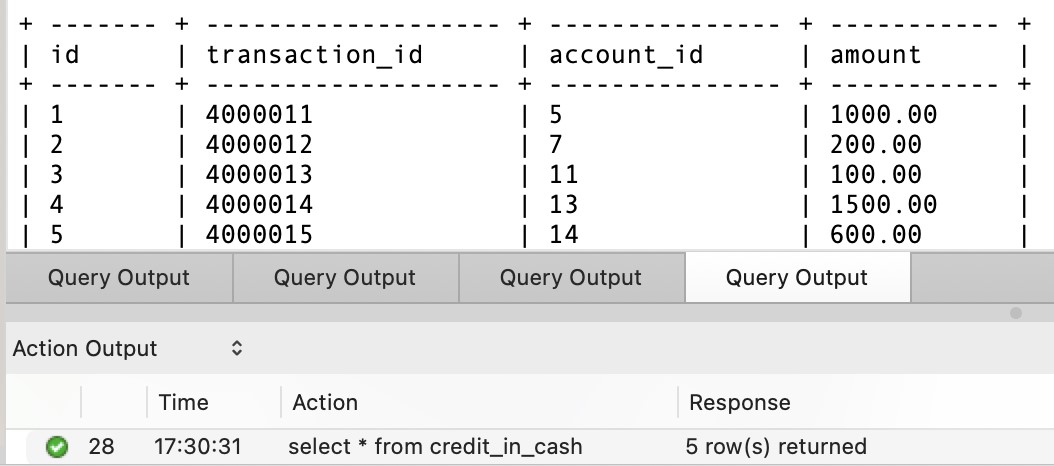
INSERT INTO TRANSACTIONS VALUES(4000011, 5, '1998-12-31', "VYDAJ", "VKLAD", 1000,20000,

INSERT INTO TRANSACTIONS VALUES(4000012, 7, '1995-12-31', "VYDAJ", "VKLAD", 200,20000, "

INSERT INTO TRANSACTIONS VALUES(4000013, 11, '1996-1-31', "VYDAJ", "VKLAD", 100,20000, "

INSERT INTO TRANSACTIONS VALUES(4000014, 13, '1998-5-12', "VYDAJ", "VKLAD", 1500,20000,

INSERT INTO TRANSACTIONS VALUES(4000015, 14, '1999-6-15', "VYDAJ", "VKLAD", 600,20000, "



**5.9) Create a table called credit\_in\_cash\_violation with 4 columns: id (stores transaction id), account\_id, amount, current\_balance (current cash credit total for account). Create a trigger called valid\_cash\_credit that inserts the cash credit transaction (operation = ‘VYKLAD’) into transactions only if the total amount of cash credit transactions (including new) for the account in credit\_in\_cash is less than or equal to 2000. If not, then insert the available balance as cash credit and insert the new amount and current total along with transaction id and account id into credit\_in\_cash\_violation. (Eg. if past transaction sum is 1700 and new transaction is 500, only insert 300 as account cannot exceed limit and insert 1700 as current\_balance and 500 as a). Insert the following rows and display credit\_in\_cash\_violation. (1 row)**

-- Create the credit\_in\_cash\_violation table if it doesn't exist

CREATE TABLE IF NOT EXISTS credit\_in\_cash\_violation ( id INT PRIMARY KEY AUTO\_INCREMENT,

account\_id INT, amount DECIMAL(10, 2), current\_balance DECIMAL(10, 2)

);

-- Create the trigger on credit\_in\_cash table

DELIMITER //

CREATE TRIGGER trg\_valid\_cash\_credit

BEFORE INSERT ON credit\_in\_cash FOR EACH ROW

BEGIN

DECLARE total\_amount DECIMAL(10, 2);

DECLARE available\_balance DECIMAL(10, 2);

-- Get the total amount of cash credit transactions (including new) for the account

SELECT COALESCE(SUM(amount), 0) INTO total\_amount

FROM credit\_in\_cash

WHERE account\_id = NEW.account\_id;

-- If total amount exceeds 2000, insert available balance as cash credit in transactions table

IF total\_amount + NEW.amount > 2000 THEN

-- Get the available balance

SELECT (2000 - total\_amount) INTO available\_balance;

-- Insert the available balance as cash credit transaction into transactions table

INSERT INTO transactions (account\_id, date, type, operation, amount, balance, k\_symbol, bank, account)

VALUES (NEW.account\_id, CURDATE(), 'type1', 'VKLAD', available\_balance,

(SELECT balance FROM accounts WHERE id = NEW.account\_id), 'k\_symbol1',

'bank1', 'account1');

-- Insert the new amount and current total into credit\_in\_cash\_violation table

INSERT INTO credit\_in\_cash\_violation (account\_id, amount, current\_balance) VALUES (NEW.account\_id, NEW.amount, total\_amount + NEW.amount);

-- Set the NEW.amount to 0 to prevent insertion of cash credit into credit\_in\_cash table

SET NEW.amount = 0;

ELSE

-- Insert the cash credit transaction into transactions table

INSERT INTO transactions (account\_id, date, type, operation, amount, balance, k\_symbol, bank, account)

VALUES (NEW.account\_id, CURDATE(), 'type1', 'VKLAD', NEW.amount, (SELECT balance FROM accounts WHERE id = NEW.account\_id), 'k\_symbol1', 'bank1',

'account1');

END IF;

END;

//

DELIMITER ;

INSERT INTO TRANSACTIONS VALUES(4000016, 5, '1998-12-31', "VYDAJ",

"VKLAD", 1500,

20000, "UROK", null, null);

INSERT INTO TRANSACTIONS VALUES(4000017, 7, '1995-12-31', "VYDAJ",

"VKLAD", 300,

20000, "UROK", null, null);

INSERT INTO TRANSACTIONS VALUES(4000018, 11, '1996-1-31', "VYDAJ",

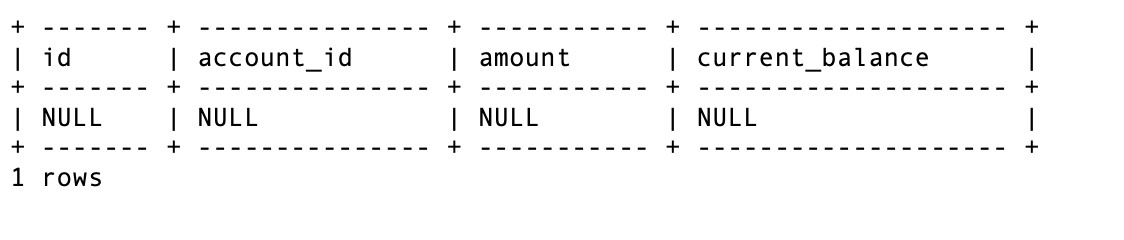
"VKLAD", 100,

20000, "UROK", null, null);

INSERT INTO TRANSACTIONS VALUES(4000019, 17, '1998-5-12', "VYDAJ",

"VKLAD", 1500,

20000, "UROK", null, null);



**5.10 ) Create a table loan\_balance with 5 columns: id (loan id), account\_id, amount, amount\_paid, amount\_left. Populate the table by adding one row for every row in loans. Copy the id, account\_id and amount, set amount\_paid to 0 and amount\_left to amount. Create a trigger that adjusts the loan\_balance every time a loan payment order is made (order.k\_symbol = ‘UVER’) by increasing the amount paid and decreasing amount left. Insert the following rows in orders and display all rows from loan\_balance. (682 rows, 4 changed)**

-- Create the loan\_balance table

CREATE TABLE loan\_balance ( id INT PRIMARY KEY,

account\_id INT, amount DECIMAL(10,2), amount\_paid DECIMAL(10,2), amount\_left DECIMAL(10,2)

);

-- Populate loan\_balance table with data from Loans table

INSERT INTO loan\_balance (id, account\_id, amount, amount\_paid, amount\_left)

SELECT id, account\_id, amount, 0, amount FROM Loans;

-- Create a trigger on Orders table

DELIMITER //

CREATE TRIGGER update\_loan\_balance\_trigger

AFTER INSERT ON Orders FOR EACH ROW

BEGIN

-- Check if the inserted order is a loan payment order with k\_symbol = 'UVER' IF NEW.k\_symbol = 'UVER' THEN

-- Update the loan\_balance table with the amount paid and amount left

UPDATE loan\_balance

SET amount\_paid = amount\_paid + NEW.amount, amount\_left = amount\_left - NEW.amount

WHERE id = NEW.account\_id;

END IF;

END;

//

DELIMITER ;

INSERT INTO ORDERS VALUES (1, 2, "AB", 12345678, 5000 , 'UVER');

INSERT INTO ORDERS VALUES (2, 25, "XY", 47299182, 100 , 'UVER');

INSERT INTO ORDERS VALUES (3, 67, "FJ", 98348264, 3000, 'UVER');

Select \* from loan\_balance

