CREATE TABLE amenities (

amenid CHAR(11) PRIMARY KEY,

lat FLOAT,

lon FLOAT,

ts TIMESTAMP,

amenity VARCHAR NOT NULL,

amname VARCHAR,

tags SUPER

);

A screenshot of a computer

Description automatically generated

CREATE TABLE customers (

custid CHAR(11) PRIMARY KEY,

custname VARCHAR,

street VARCHAR,

city VARCHAR,

province CHAR(2),

postalcode CHAR(6)

);

A screenshot of a phone

Description automatically generated

CREATE TABLE paymentmethods (

pmid CHAR(9) PRIMARY KEY,

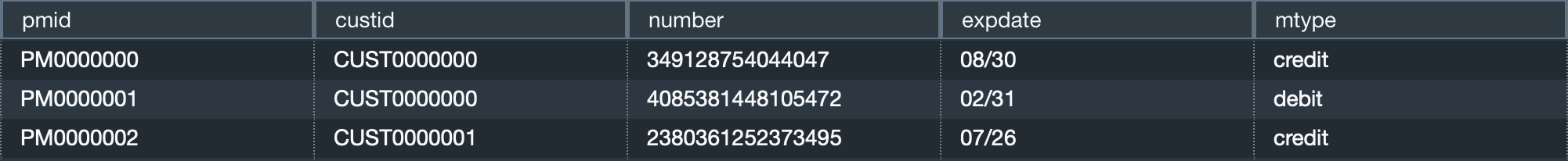
custid CHAR(11) REFERENCES customers(custid),

number VARCHAR(20),

expdate CHAR(5),

mtype VARCHAR(6)

);



CREATE TABLE purchases (

purchid CHAR(15) PRIMARY KEY,

custid CHAR(11) REFERENCES customers(custid),

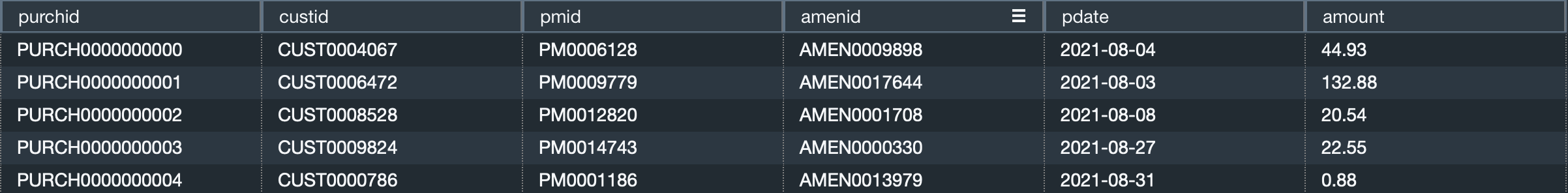
pmid CHAR(9) REFERENCES paymentmethods(pmid),

amenid CHAR(11) REFERENCES amenities(amenid),

pdate DATE,

amount DECIMAL(7,2)

);



CREATE TABLE greater\_vancouver\_prefixes (

city VARCHAR(15),

pcprefix CHAR(3)

);

A screenshot of a computer

Description automatically generated

#### QUESTION 2: WHO SPENDS MORE OVERALL?

Our second question will require more work to answer:

Consider the three groups of people: people who live in the Vancouver region, visitors from other BC areas, and visitors from outside BC altogether. Which group spent the most per transaction?

To answer this, write a query that returns the following table,

From\_BC\_non\_Van From\_Van Count Average Median

...

Tables needed: purchases, customers, greater\_vancouver\_prefixes

CREATE VIEW vancouver\_custs AS

WITH

vprefixes (vp) AS

(SELECT DISTINCT pcprefix FROM greater\_vancouver\_prefixes)

SELECT

custid,

CASE

WHEN SUBSTRING(customers.postalcode, 1, 3) IN (SELECT vp FROM vprefixes)

THEN 1

ELSE 0

END AS in\_vancouver

FROM customers;

A screenshot of a computer

Description automatically generated

DROP VIEW IF EXISTS vancouver\_custs CASCADE;

CREATE VIEW vancouver\_custs AS

WITH

vprefixes (vp) AS

(SELECT DISTINCT pcprefix FROM greater\_vancouver\_prefixes)

SELECT

custid,

CASE

WHEN SUBSTRING(customers.postalcode, 1, 3) IN (SELECT vp FROM vprefixes)

THEN 1

ELSE 0

END AS in\_vancouver

FROM customers;

DROP VIEW IF EXISTS customers\_in\_van CASCADE;

CREATE VIEW customers\_in\_van AS

SELECT

customers.custid,

CASE

WHEN vancouver\_custs.in\_vancouver = 0 AND customers.province='BC'

THEN true

ELSE false

END AS From\_BC\_non\_Van,

CASE

WHEN vancouver\_custs.in\_vancouver = 1

THEN true

ELSE false

END AS From\_Van

FROM customers

JOIN vancouver\_custs ON vancouver\_custs.custid = customers.custid;

SELECT

customers\_in\_van.From\_BC\_non\_Van,

customers\_in\_van.From\_Van,

COUNT(customers.custid),

AVG(purchases.amount),

MEDIAN(purchases.amount)

FROM customers

JOIN vancouver\_custs ON vancouver\_custs.custid = customers.custid

JOIN purchases ON purchases.custid = customers.custid

JOIN customers\_in\_van ON customers\_in\_van.custid = customers.custid

GROUP BY customers\_in\_van.From\_BC\_non\_Van, customers\_in\_van.From\_Van

ORDER BY MEDIAN(purchases.amount) asc

Or

DROP VIEW IF EXISTS vancouver\_custs CASCADE;

CREATE VIEW vancouver\_custs AS

WITH

vprefixes (vp) AS

(SELECT DISTINCT pcprefix FROM greater\_vancouver\_prefixes)

SELECT

custid,

CASE

WHEN SUBSTRING(customers.postalcode, 1, 3) IN (SELECT vp FROM vprefixes)

THEN 1

ELSE 0

END AS in\_vancouver

FROM customers;

SELECT

(vancouver\_custs.in\_vancouver = 0 AND customers.province = 'BC') AS From\_BC\_non\_Van,

(vancouver\_custs.in\_vancouver = 1) AS From\_Van,

COUNT(customers.custid) AS cust\_count,

AVG(purchases.amount),

MEDIAN(purchases.amount)

FROM customers

JOIN vancouver\_custs ON vancouver\_custs.custid = customers.custid

JOIN purchases ON purchases.custid = customers.custid

GROUP BY (vancouver\_custs.in\_vancouver = 0 AND customers.province = 'BC'),

(vancouver\_custs.in\_vancouver = 1)

ORDER BY MEDIAN(purchases.amount) asc;

from\_bc\_non\_van from\_van cust\_count avg median

false true 10384 86.01 27.37

true false 3899 95.16 30.08

false false 15717 112.89 33.27

QUESTION 3: WHO SPENDS MORE ON SUSHI?

Our third question:

Who spends more at restaurants that serve sushi: locals (residents of Greater Vancouver) or tourists?

Who 🡪 table:

CREATE TABLE customers (

custid CHAR(11) PRIMARY KEY,

custname VARCHAR,

street VARCHAR,

city VARCHAR,

province CHAR(2),

postalcode CHAR(6)

);

Spends more 🡪 table:

CREATE TABLE purchases (

purchid CHAR(15) PRIMARY KEY,

custid CHAR(11) REFERENCES customers(custid),

pmid CHAR(9) REFERENCES paymentmethods(pmid),

amenid CHAR(11) REFERENCES amenities(amenid),

pdate DATE,

amount DECIMAL(7,2)

);

Restaurants 🡪 table:

CREATE TABLE amenities (

amenid CHAR(11) PRIMARY KEY,

lat FLOAT,

lon FLOAT,

ts TIMESTAMP,

amenity VARCHAR NOT NULL,

amname VARCHAR,

tags SUPER

);

Locals or tourists 🡪 table:

DROP VIEW IF EXISTS vancouver\_custs CASCADE;

CREATE VIEW vancouver\_custs AS

WITH

vprefixes (vp) AS

(SELECT DISTINCT pcprefix FROM greater\_vancouver\_prefixes)

SELECT

custid,

CASE

WHEN SUBSTRING(customers.postalcode, 1, 3) IN (SELECT vp FROM vprefixes)

THEN 1

ELSE 0

END AS in\_vancouver

FROM customers;

WITH sushi (amenid) AS

(SELECT amenities.amenid

FROM amenities

WHERE tags.cuisine IS NOT NULL AND (tags.cuisine ILIKE '%sushi%' OR tags.cuisine ILIKE 'udon'))

SELECT

AVG(purchases.amount) AS avg,

MEDIAN(purchases.amount) AS median,

in\_vancouver

FROM purchases

JOIN amenities ON amenities.amenid = purchases.amenid

JOIN customers ON customers.custid = purchases.custid

JOIN vancouver\_custs ON vancouver\_custs.custid = customers.custid AND vancouver\_custs.custid = purchases.custid

WHERE amenities.amenity='restaurant' AND amenities.amenid IN (SELECT amenid FROM sushi)

GROUP BY vancouver\_custs.in\_vancouver

ORDER BY vancouver\_custs.in\_vancouver;

Seems to be working!

avg median in\_vancouver

85.8 88.07 0

77.57 78.8 1

### **Question 4: Average purchase per day for the first five days?**

Our final question:

What was the average purchase per day for the first five days of August?

To answer this, write a query that returns the following two-column, five-line table:

pdate avg

2021-08-01 ...

2021-08-02 ...

2021-08-03 ...

2021-08-04 ...

2021-08-05 ...

Order the table by increasing date. Use the [Redshift DATE\_PART function](https://docs.aws.amazon.com/redshift/latest/dg/r_DATE_PART_function.html) to extract the month and day from the pdate field.

We're going to use this query to highlight the massive parallelism potentially available to Spectrum when it reads tables directly from the S3 data lake.

SELECT pdate, AVG(amount)

FROM purchases

WHERE DATE\_PART(day, purchases.pdate) IN (1, 2, 3, 4 ,5) AND DATE\_PART(month, purchases.pdate)=08

GROUP BY purchases.pdate

ORDER BY purchases.pdate asc;

#### DATA READ BY REDSHIFT FROM AN INTERNAL TABLE

Run the query first as a regular Redshift query on the same purchases table you used in Part 1. The results themselves aren't as interesting as how the query is executed by the two query engines.

From the Redshift main page, select "Queries and loads", sort queries by decreasing start time, find the previous query, and select the "query plan" tab. The only part of the plan of interest to us is the bottom row, "Seq Scan on purchases". From that row, note the number of bytes and number of rows that Redshift scanned.

Number of bytes: 94.06 KB

Number of rows: 4703

SELECT pdate, AVG(amount)

FROM s3ext.purchases

WHERE DATE\_PART(day, pdate) IN (1, 2, 3, 4 ,5) AND DATE\_PART(month, pdate)=08

GROUP BY pdate

ORDER BY pdate asc;

Number of bytes: 120 bytes

Number of rows: 5

This is the results of the scan, passed back to Redshift by Spectrum.

It should be obvious the real work for this query was not performed by Redshift but rather Spectrum. And to see how much data Spectrum read , execute the following query of Redshift's internal management table, pg\_catalog.svl\_s3query\_summary:

SELECT

external\_table\_name,

s3\_scanned\_bytes,

s3\_scanned\_rows,

avg\_request\_parallelism,

max\_request\_parallelism

FROM pg\_catalog.svl\_s3query\_summary

ORDER BY starttime DESC

LIMIT 1;

First note the number of bytes and number of rows Spectrum read.

external\_table\_name s3\_scanned\_bytes s3\_scanned\_rows avg\_request\_parallelism max\_request\_parallelism

S3 Subquery dev\_s3ext\_purchases 267396 4703 1.5 5

Num Bytes: 267396 Num Rows: 4703

Also consider the content of the S3 bucket. Navigate your browser to [s3://sfu-cmpt-732-redshift](https://s3.console.aws.amazon.com/s3/buckets/sfu-cmpt-732-redshift?region=us-west-2&tab=objects).

What we see:

1. The external “date lake” version of purchases (ext\_purchases) contains a single folder *for each day* with each of the purchases made that day.
2. The original “data warehouse” version of purchases contains a single csv file containing all of the purchases made that month.

The second thing to note is the two parallelism values. The external purchase table is laid out so that every day can be processed in parallel by a separate Spectrum instance. To see a more extreme case, rerun the Spectrum query for all 31 days of the month. Now rerun the query on svl\_s3query\_summary but with LIMIT 2, so you can compare the statistics for the 5-day and 31-day queries. What was the maximum parallelism you got on this larger query?

external\_table\_name s3\_scanned\_bytes s3\_scanned\_rows avg\_request\_parallelism max\_request\_parallelism

S3 Subquery dev\_s3ext\_purchases 1705186 30000 4.3 10

S3 Subquery dev\_s3ext\_purchases 267396 4703 1.5 5