THE UNIVERSITY OF SYDNEY

School of Computer Science
Uwe Roehm

DATA2001/2901: Data Science, Big Data, and Data Diversity **Semester 1, 2021**

Tutorial Week 5: Scalable Data Analytics

In this tutorial you will learn about the role of indexing and database statistics for fast data analytics on large data sets. We are using a new scenario in this tutorial of data from some car hire company which we would like to analyse.

System Requirements You will need the following tools for this tutorial:

- The University's Cisco VPN client to access our internal PostgreSQL server. Also you will also need the FortiClient VPN client if accessing from China.
- pgAdmin4 to connect to PostgreSQL.
- Login to `soitpw11d59.shared.sydney.edu.au` with your Postgresql login (`y21s1d2x01_`*`unikey`* and your SID as password - unless you had it changed).

Common Errors A list of common errors and solutions has been provided on Ed https://edstem.org/courses/5592/discussion/399215. Please have a look at this post (and you can add to it!) for common errors that we are all likely experiencing.

Exercise 1. Review: Storage Layer of a Database System

Start by discussing the following review questions about data on external storage in a database management system (DBMS):

1. Why does a DBMS store data on external storage?

   Solution: A DBMS stores data on external storage because the quantity of data is vast, and must persist across many executions. You can't hold it all in a volatile (non-persistant) storage such as memory.

2. Why are I/O costs important in a DBMS?

   Solution: I/O costs are of primary important to a DBMS because these costs typically dominate the time it takes to run most database operations. Optimizing the amount of I/O's for an operation can result in a substantial increase in speed in the time it takes to run that operation

Exercise 2. System Catalog Exploration

What information is stored in the database system catalogs[1]? Explore which tables you have in your local schema and which indexes are already in place, for example by issuing the following SQL command in pgAdmin

```
SELECT * FROM pg_indexes
```

[1] Online documentation at: https://www.postgresql.org/docs/12/catalogs.html

You can also explore the schema and statistics via the sidebar of pgAdmin4 by selecting a table inside a schema, and then looking at the 'Statistics' and the 'Dependents' tabs on the right of the window (you might need to 'refresh' the sidebar after you created some new table or index).

Note how the system catalog includes statistics about the size of tables and even value distributions of attributes. These statistics are only infrequent updated by PostgreSQL though. To tell PostgreSQL explicitly to clean up and update its statistics after some updates, you can issue this command when needed: `VACUUM ANALYZE`

Exercise 3. Indexing Tuning with PostgreSQL

Download and run the SQL statements in the file `DATA2001_Tutorial_Wk5.sql`. This will create three relations (`Driver`, `Vehicle`, and `Triplog`) and some demo data for a car hire company which logs every trip by a driver.

(a) Check whether there are already some indexes available on your schema. To do so, issue the SELECT statement on PG_INDEXES as given above, and then explore the details on columns and indexes shown in the result. Can you explain, what you see?

Solution: get the students to try

```
select * from pg_indexes where schemaname = 'waterinfo';
```

This should yield some familiar table names

(b) Test then the following queries, recording the execution times:

```
SELECT * FROM Driver WHERE given_name = 'Eric';
SELECT * FROM Driver WHERE family_name = 'Andrews';
SELECT COUNT(*) FROM Vehicle WHERE year = 2005;
SELECT COUNT(*) FROM Vehicle WHERE year < 2005;
SELECT * FROM TripLog WHERE distance > 200;
SELECT * FROM TripLog WHERE distance > 200 AND distance < 250;
SELECT * FROM TripLog WHERE distance > 200
                      AND (end_time-start_time) < (INTERVAL '1 hours');
```

(c) Try these a few times to see how the execution time varies. Why is it not the same each time?

(d) Do all queries take about the same time? If not, why?
Hint: You can get data on how the queries are executed by putting `EXPLAIN ANALYZE` before the rest of the query.

(e) Create the following indexes for the TripLog relation for the *distance*, *car_id*, *vehicle_id* and *year* attributes:

```
CREATE INDEX distance_triplog_ind ON TripLog (distance);
CREATE INDEX car_id_triplog_ind ON TripLog (car_id);
CREATE INDEX driver_id_triplog_ind ON TripLog (driver_id);
CREATE INDEX vehicle_year_ind ON Vehicle (year);
```

(f) Rerun the above queries with the indexes and compare the estimates to see if the query is affected. (You may find it useful to run `VACUUM ANALYZE` to update the databases' statistics.)

Exercise 4. Scalable Data Analysis with SQL

The Federal Government has set an Australia-wide speed limit of 110 km/h. You have been hired to find out who exceeded this limit during their trip. Consider the following questions:

(a) How many vehicles went over the maximum speed during their trip?

Solution: This is an example query, however individual results may vary due to the random() function in the data creation:

```
SELECT COUNT(*)
FROM TripLog t
WHERE (t.distance/((EXTRACT(epoch FROM (t.end_time-t.start_time)))/3600)) > 110;
```

(b) What is the minimal number tables to find this information?

Solution: `TripLog` has all the information relating to start times, end times and distance travelled.

(c) We wish to send a penalty letter to all the drivers that violated the maximum speed. Construct a query to give the addresses of drivers who exceeded the speed limit. Your result should contain, driver address, given and family names, vehicle model and description, trip start time and speed.

Solution:

```
SELECT
    d.given_name,
    d.family_name,
    d.address,
    v.model,
    v.descr as description,
    t.start_time,
    (t.distance/((EXTRACT(epoch FROM (t.end_time-t.start_time)))/3600)) as speed
FROM
    (Driver d INNER JOIN TripLog t ON d.driver_id = t.driver_id)
              INNER JOIN Vehicle v ON t.car_id = v.car_id
WHERE
    (t.distance/((EXTRACT(epoch FROM (t.end_time-t.start_time)))/3600)) > 110
```

(d) How many tables must you access?

Solution: Speed details can be found in the Trip Log table; names and address can be found in the Driver table; model and descriptions can be found in the Vehicle table.

(e) How many joins are required?

Solution: Two joins between tables `Driver` and `Vehicle` to `TripLog`.

(f) What types of indexes are required?

Solution: We have a range query on an operation that fetches 3 columns from the `TripLog` table. So we can create an index based on this operation on *distance*, *start_time* and *end_time*.

We also have a two joins in the ID values for `Vehicle` and `Driver` on `TripLog`, so an index on these values should speed up the join operation.

(g) On which tables and on which columns should these indexes be built?

Solution: We can create an index on *car_id* in `Vehicle` and `TripLog` as well as *driver_id* in `Driver` and `TripLog`. This would increase the speed of the join. We can create an index for the `TripLog` operation on *end_time*, *start_time* and *distance*. E.g. :

```
CREATE INDEX driver_driver_id_idx ON Driver(driver_id);
CREATE INDEX triplog_driver_id_idx ON TripLog(driver_id);
CREATE INDEX vehicle_car_id_idx ON Vehicle(car_id);
CREATE INDEX triplog_car_id_idx ON TripLog(car_id);
CREATE INDEX triplog_speed_expression_idx on
        TripLog((distance/((EXTRACT(epoch FROM (end_time-start_time)))/3600)));
```

Exercise 5. SQL Online Tutorial in Grok

Please continue going through the SQL tutorial in Grok in your own time and finish all eight modules by the Easter break as preparation for the (online) SQL quiz in Week 7.

Exercise 6. Advanced Task (DATA2901)

Let's do some advanced analysis of the type of drivers based on the trip data:

(a) Write an SQL window query that finds the top-10 drivers in terms of number of trips. List each driver with *driver_id*, *count* of trips, and their (dense) *rank*.

Solution:

```
SELECT driver_id, count, DENSE_RANK() OVER (ORDER BY count DESC) AS rank
  FROM ( SELECT driver_id, COUNT(*) AS count
           FROM Driver JOIN TripLog USING (driver_id)
          GROUP BY driver_id
       ) AS DriverStats
 ORDER BY rank
 LIMIT 10;
```

(b) Extend your query from (a) to also list and (densely) rank the total distance travelled by each driver. Still restrict the list to the top-10 drivers with regard to trip count.

Solution:

```
SELECT driver_id, count, DENSE_RANK() OVER (ORDER BY count DESC) AS rankCount,
                sumdist, DENSE_RANK() OVER (ORDER BY sumdist DESC) AS rankDist
  FROM (SELECT driver_id, COUNT(*) AS count, SUM(distance) AS sumdist
          FROM Driver JOIN TripLog USING (driver_id)
       GROUP BY driver_id
       ) AS DriverStats
 ORDER BY rankCount
 LIMIT 10;
```

(c) Extend your query from (b) to also categorise the drivers whether they are more a 'weekender' or a 'worker' driver. A weekend driver does at least half of his or her trips during a weekend (Sat or Sun), while a workday driver does most trips on a working day. Tip: Have a look into the CASE statement for this.

Solution:

```
SELECT driver_id, count, DENSE_RANK() OVER (ORDER BY count DESC) AS rank,
                sumdist, DENSE_RANK() OVER (ORDER BY sumdist DESC) AS rankDist,
```

```
               CASE
                  WHEN weekendtrips >= workdaytrips THEN 'weekender'
                  ELSE 'worker'
               END
FROM (SELECT driver_id, COUNT(*) AS count, SUM(distance) AS sumdist
      SUM(CASE WHEN EXTRACT(dow FROM start_time) BETWEEN 1 AND 5 THEN 1 ELSE 0 END) AS workdaytrips,
      SUM(CASE WHEN EXTRACT(dow FROM start_time) BETWEEN 1 AND 5 THEN 0 ELSE 1 END) AS weekendtrips
        FROM Driver JOIN TripLog USING (driver_id)
       GROUP BY driver_id
     ) AS DriverStats
ORDER BY rank
LIMIT 10;
```

(d) Discuss how partitioning (horizontal vs vertical) the above car hire company data could help in scalability and accessibility of the data.