Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Лабораторна робота №1

Програмування інтелектуальних інформаційних систем

Тема: Columnar table storage

Виконав
студент групи ІП-12:
Горобець О. С.

Перевірив:
Баришич Л. М.

Київ 2023

Лабораторна робота №1

1. Setting up the environment

- PostgreSQL was chosen as the main RDBMS
- CitusData as an open source extension to Postgres that distributes data and queries across multiple nodes in a cluster. Also added CitusManager to control the load distribution.
- Docker-compose was used to start up all the services.

Docker-compose file:

```yaml
version: '3.1'

services:
  master:
    container_name: "${COMPOSE_PROJECT_NAME:-citus}_master"
    image: "citusdata/citus:12.0.0"
    ports: ["${COORDINATOR_EXTERNAL_PORT:-5432}:5432"]
    labels: ["com.citusdata.role=Master"]
    environment: &AUTH
      POSTGRES_USER: "${POSTGRES_USER:-postgres}"
      POSTGRES_PASSWORD: "${POSTGRES_PASSWORD:-postgrespass}"
      PGUSER: "${POSTGRES_USER:-postgres}"
      PGPASSWORD: "${POSTGRES_PASSWORD:-postgres}"
      POSTGRES_HOST_AUTH_METHOD: "${POSTGRES_HOST_AUTH_METHOD:-trust}"
    volumes:
      - /resources:/data
  worker:
    image: "citusdata/citus:12.0.0"
    labels: ["com.citusdata.role=Worker"]
    depends_on: [manager]
    environment: *AUTH
    command: "/wait-for-manager.sh"
    volumes:
      - healthcheck-volume:/healthcheck
  manager:
    container_name: "${COMPOSE_PROJECT_NAME:-citus}_manager"
    image: "citusdata/membership-manager:0.3.0"
    volumes:
      - "${DOCKER_SOCK:-/var/run/docker.sock}:/var/run/docker.sock"
      - healthcheck-volume:/healthcheck
    depends_on: [master]
    environment: *AUTH
volumes:
  healthcheck-volume:
```

Docker-compose up and running:

piic		Running (3/3)	1.95%		2 hours ago		
worker-1 ee8496c73a8f	citusdata/citus:12.0.(	Running	0%		2 hours ago		
piic_manage b0913c345f60	citusdata/membershi	Running	1.88%		2 hours ago		
piic_master a475f561bd08	citusdata/citus:12.0.(	Running	0.07%	5432:5432	2 hours ago		

2. Creation of row-based db

```
DROP DATABASE IF EXISTS flights_db;

CREATE DATABASE flights_db;

\c  flights_db;

CREATE TABLE airlines
(
    iata_code varchar(2),
    airline   varchar(30),

    CONSTRAINT PK_airlines PRIMARY KEY (iata_code)
);

CREATE TABLE airports
(
    iata_code varchar(3),
    airport   varchar(80),
    city      varchar(30),
    state     varchar(2),
    country   varchar(30),
    latitude  decimal(11, 4),
    longitude decimal(11, 4),

    CONSTRAINT PK_airports PRIMARY KEY (iata_code)
);

CREATE TABLE flights
(
    year                 smallint,
    month                smallint,
    day                  smallint,
    day_of_week          smallint,
    fl_date              date,
    carrier              varchar(2),
    tail_num             varchar(6),
    fl_num               smallint,
    origin               varchar(5),
    dest                 varchar(5),
    crs_dep_time         varchar(4),
    dep_time             varchar(4),
    dep_delay            decimal(13, 2),
    taxi_out             decimal(13, 2),
    wheels_off           varchar(4),
    wheels_on            varchar(4),
    taxi_in              decimal(13, 2),
    crs_arr_time         varchar(4),
    arr_time             varchar(4),
    arr_delay            decimal(13, 2),
    cancelled            decimal(13, 2),
```

```
    cancellation_code   varchar(20),
    diverted            decimal(13, 2),
    crs_elapsed_time    decimal(13, 2),
    actual_elapsed_time decimal(13, 2),
    air_time            decimal(13, 2),
    distance            decimal(13, 2),
    carrier_delay       decimal(13, 2),
    weather_delay       decimal(13, 2),
    nas_delay           decimal(13, 2),
    security_delay      decimal(13, 2),
    late_aircraft_delay decimal(13, 2)
);


CREATE INDEX idx_flights_year ON flights (year);
CREATE INDEX idx_flights_carrier ON flights (carrier);
CREATE INDEX idx_flights_carrier_delay ON flights (carrier_delay);
CREATE INDEX idx_flights_weather_delay ON flights (weather_delay);
CREATE INDEX idx_flights_nas_delay ON flights (nas_delay);
CREATE INDEX idx_flights_security_delay ON flights (security_delay);
CREATE INDEX idx_flights_late_aircraft_delay ON flights (late_aircraft_delay);
CREATE INDEX idx_flights_arr_delay ON flights (arr_delay);
CREATE INDEX idx_flights_month ON flights (month);
CREATE INDEX idx_flights_dest ON flights (dest);
```

3. Creation of columnar db

```
DROP DATABASE IF EXISTS flights_column_db;

CREATE DATABASE flights_column_db;

\c columnstore_bts;

CREATE EXTENSION IF NOT EXISTS citus;

DROP TABLE IF EXISTS airlines;
DROP TABLE IF EXISTS airports;
DROP TABLE IF EXISTS flights;


CREATE TABLE airlines
(
    iata_code varchar(2),
    airline   varchar(30),
    CONSTRAINT PK_airlines PRIMARY KEY (iata_code)
) USING columnar;

CREATE TABLE airports
(
    iata_code varchar(3),
    airport   varchar(80),
    city      varchar(30),
    state     varchar(2),
    country   varchar(30),
    latitude  decimal(11, 4),
    longitude decimal(11, 4),
    CONSTRAINT PK_airports PRIMARY KEY (iata_code)
) USING columnar;

CREATE TABLE flights
(
```

```
    year                  smallint,
    month                 smallint,
    day                   smallint,
    day_of_week           smallint,
    fl_date               date,
    carrier               varchar(2),
    tail_num              varchar(6),
    fl_num                smallint,
    origin                varchar(5),
    dest                  varchar(5),
    crs_dep_time          varchar(4),
    dep_time              varchar(4),
    dep_delay             decimal(13, 2),
    taxi_out              decimal(13, 2),
    wheels_off            varchar(4),
    wheels_on             varchar(4),
    taxi_in               decimal(13, 2),
    crs_arr_time          varchar(4),
    arr_time              varchar(4),
    arr_delay             decimal(13, 2),
    cancelled             decimal(13, 2),
    cancellation_code     varchar(20),
    diverted              decimal(13, 2),
    crs_elapsed_time      decimal(13, 2),
    actual_elapsed_time   decimal(13, 2),
    air_time              decimal(13, 2),
    distance              decimal(13, 2),
    carrier_delay         decimal(13, 2),
    weather_delay         decimal(13, 2),
    nas_delay             decimal(13, 2),
    security_delay        decimal(13, 2),
    late_aircraft_delay   decimal(13, 2)
) USING columnar;
```

4. Seed data import

```
COPY airlines
    FROM '/data/resources/airlines.csv'
    DELIMITER ','
    CSV HEADER;

COPY airports
    FROM '/data/resources/airports.csv'
    DELIMITER ','
    CSV HEADER;

COPY flights
    FROM '/data/resources/flights.csv'
    DELIMITER ','
    CSV HEADER;
```

5. Queries

```
-- 1. Count summary delay for each city
-- Execution time:
-- row: 1s 271ms
-- columnar: 570 ms
SELECT C.city, A.departure_delay, B.arrival_delay, (A.departure_delay +
B.arrival_delay) as total_delay
FROM (SELECT flights.origin as city, sum(flights.dep_delay) as departure_delay
FROM flights GROUP BY city) AS A
         FULL OUTER JOIN
```

```
    (SELECT flights.dest as city, sum(flights.arr_delay) as arrival_delay FROM
flights GROUP BY city) AS B
     ON A.city = B.city
         INNER JOIN airports AS C ON A.city = C.iata_code OR B.city =
C.iata_code;


-- 2. Count the number of flights arriving and departing for each city
-- Execution time:
-- row: 523ms
-- columnar: 445 ms
SELECT C.city, origin_count, dest_count, (origin_count + dest_count) as
overall_count
FROM (SELECT COUNT(*) as origin_count, origin as city FROM flights GROUP BY
flights.origin) AS A
         FULL OUTER JOIN
    (SELECT COUNT(*) as dest_count, dest as city FROM flights GROUP BY
flights.dest) AS B
     ON A.city = B.city
         INNER JOIN airports AS C ON A.city = C.iata_code OR B.city =
C.iata_code;


-- 3. Find city with the minimal delay. Delay is counted as a sum of "arr_delay"
and "dep_delay"
-- Execution time:
-- row: 450 ms
-- columnar: 360 ms
SELECT B.city, MIN(delay) as min_delay
FROM (SELECT MIN(flights.arr_delay + flights.dep_delay) as delay, dest as city
      FROM flights
      GROUP BY dest) as A
         INNER JOIN airports as B
                    ON A.city = B.iata_code
GROUP BY B.city
ORDER BY min_delay
LIMIT 1;

-- Execution time:
-- row: 464 ms
-- columnar: 384 ms
SELECT B.city, MAX(delay) as min_delay
FROM (SELECT MAX(flights.arr_delay + flights.dep_delay) as delay, dest as city
      FROM flights
      GROUP BY dest) as A
         INNER JOIN airports as B
                    ON A.city = B.iata_code
GROUP BY B.city
ORDER BY min_delay
LIMIT 1;


-- 4. Find all flights with a delay greater than the average delay time
-- Execution time:
-- row: 464 ms
-- columnar: 320 ms
SELECT *
FROM flights
WHERE flights.arr_delay + flights.dep_delay > (SELECT AVG(flights.dep_delay +
flights.arr_delay) as avg_delay
                                               FROM flights);
```

Lets run an additional query to check the memory taken by each database:

```
SELECT pg_size_pretty(pg_database_size('flights_column_db')) AS
columnstore_size,
       pg_size_pretty(pg_database_size('flights_db'))        as rowstore_size;
```

Results:

	columnstore_size	rowstore_size
1	37 MB	313 MB

6. Outcomes:

The observed data indicates that, on average, columnar databases exhibit a roughly 1.5-fold improvement in execution time compared to row-based databases. It is important to note that these tests possess some limitations in terms of accuracy, primarily due to their failure to account for variables such as system load variations during query execution, stemming from the absence of an isolated environment.

Nevertheless, an obvious trend emerges from the results, highlighting the overall efficiency advantage of columnar databases in both data retrieval and storage. Specifically, the columnar database demonstrates a substantial advantage in data storage efficiency, with a size of approximately 37 megabytes, compared to the considerably larger 313 megabytes required for row-based storage.

Висновок

У ході лабораторної роботи ознайомилися з бібліотекою pandas. На практиці застосували методи колекції Series та DataFrame. Розглянули різні способи модифікації та відображення даних. Проаналізували дані з набору даних катастрофи «Титаніка», знайшли наймолодших та найстарших пасажирів. Побудували гістограму розподілу віку пасажирів величин. На практиці використали сортування та фільтрацію по певним критеріям.