Advanced Databases

Assignment 10 - Columnar store (spec.)

Group: A2
Mustafa Tayyip BAYRAM 257639
Furkan ÖCALAN 257638

Where and Why Should We Use Columnar Storage?

Row-oriented databases store each record in one or more contiguous blocks on disk. Column-oriented databases store each column in one or more contiguous blocks. Each scheme is better suited to different use cases, as the following example illustrates.

As you know our database holding too much data that we have not searched for . Consider Customers table:

CUSTOMER_ID	FIRST_NAME	LAST_NAME	PHONE	EMAIL	STREET	CITY	STATE	ZIP_CODE
1	Ynes	Eberdt	952-875-8729	yeberdt0@seesaa.net	3 School Parkway	Litvínovice	EMB	370 01
2	Evangeline	Francke	465-749-5215	efranckel@europa.eu	1950 Sommers Drive	Loja	RF	9604
3	Garnette	Larcombe	357-875-8513	glarcombe2@soundcloud.com	32 Ryan Alley	As Saffānīyah	ERZ	196581

Businesses handle transactions using online transaction-processing (OLTP) software. All the fields in each row are important, so for OLTP it makes sense to store items on disk by row, with each field adjacent to the next in the same block on the hard drive:

Ynes, Eberdt, 952-875-8729, yeberdt0@seesaa.net, 3 School Parkway, Litvínovice, EMB, 370, 01

Evangeline, Francke, 465-749-5215,efrancke1@europa.eu, 1950 Sommers Drive, Loja, RF, 9604

Garnette, Larcombe, 357-875-8513, glarcombe2@soundcloud.com, 32 Ryan Alley As Saffānīyah, ERZ, 196581

Some of the information from transactions is useful to inform business decisions, What's called online analytical processing (OLAP). For instance, we might want see how price affects sales, or to zero in on the referrers that send it the most traffic so it can determine where to advertise. For queries like these, we do not care about row-by-row values, but rather the information in certain columns for all rows.

For OLAP purposes, it's better to store information in a columnar database, where blocks on the disk might look like:

Ynes, Evangeline, Garnette

Eberdt, Francke, Larcombe

952-875-8729, 465-749-5215, 357-875-8513

yeberdt0@seesaa.net, efrancke1@europa.eu, glarcombe2@soundcloud.com

3 School Parkway, 1950 Sommers Drive, , 32 Ryan Alley

Litvínovice , Loja, As Saffānīyah

EMB, RF, ERZ

370, 01 , 9604, 196581

Columnar storage lets us ignore all the data that does not apply to a particular query, because we can retrieve the information from just the columns we want. By contrast, if we were working with a row-oriented database and we wanted to know, say, the average population density in cities with more than 120000 people, our query would access each record in the database (meaning all of its fields) to get the information from the two columns whose data you needed, which would involve a lot of unnecessary disk seeks and disk reads, which also impact performance.

Which tables/columns will be stored in columnar store?

- **Products Table => list_price and model_year columns**

1. QUERY

```
SELECT product_id,brand_name ,product_name, model_year, list_price, category_name FROM

    (SELECT * FROM
        (SELECT * FROM
          ( SELECT * from MUTABAY.products_range prod_outer
              where 1 = (
                  SELECT COUNT(Distinct list_price)
                  FROM MUTABAY.products_range prod_inner
                  WHERE prod_outer.brand_id = prod_inner.brand_id
                  AND prod_outer.list_price < prod_inner.list_price
                  )
          ) prod
          FULL OUTER JOIN
          MUTABAY.brands_hash brands on brands.brand_id = prod.brand_id
        ) prod_brand
        FULL OUTER JOIN
        MUTABAY.categories categories on categories.category_id = prod_brand.category_id
    )prod_brand_cat

where list_price > 990000 AND model_year < 2020

GROUP BY product_id,brand_name ,product_name, model_year, list_price, category_name
ORDER BY product_id ASC;
```

4. QUERY

```
update MUTABAY.products products
set model_year =
                        (
      select distinct(product_id) as total_product
      from MUTABAY.order_items order_items
        full outer join MUTABAY.orders orders on
          (orders.order_id = order_items.order_id)
        full outer join MUTABAY.stores stores on
          (orders.store_id = stores.store_id)
        where stores.store_id =
          (
          select store_id from MUTABAY.stocks
            full outer join MUTABAY.products products on
              (stocks.product_id = products.product_id)
                where ((model_year between 2020 and 1958) or (ROUND(list_price) < 990.000))
                    or UPPER ( SUBSTR(product_name,2,3 ) )LIKE 'D%'
                fetch first 1 rows only
          )
      fetch first 1 rows only
                        );
```

- **Order_Items Table => discount column and Products Table => list_price**

1.QUERY

```
SELECT products.product_id, products.product_name, products.list_price,
    orders.order_date, orders.required_date, orders.order_status,
    categories.category_name, brands.brand_name, quantity_id,
    discount,COUNT(quantity_id) quantity_count, (quantity_id * discount * products.list_price) total
FROM MUTABAY.order_items order_items
  full outer join MUTABAY.orders orders on
    (orders.order_id = order_items.order_id)
  full outer join MUTABAY.products products on
    (products.product_id = order_items.product_id)
  full outer join MUTABAY.brands brands on
    (brands.brand_id = products.brand_id)
  full outer join MUTABAY.categories categories on
    (categories.category_id = products.category_id)
  full outer join MUTABAY.staffs staffs on
    (staffs.staff_id = orders.staff_id)
  WHERE (shipped_date - order_date ) > 2
    OR
    (shipped_date - order_date ) = 0
    OR
    (shipped_date - order_date ) < 0

GROUP BY products.product_id, products.product_name, products.list_price,
    orders.order_date, orders.required_date, orders.order_status,
    categories.category_name, brands.brand_name, quantity_id,
    discount
having AVG(list_price) > 10000
Order by order_status;
```

- **Staffs Table => salary column**

1. QUERY

```
SELECT first_name,last_name, active, salary, stores.store_name, stores.city, stores.state
FROM
(
  SELECT staffs.*,avg(salary) over (partition by store_id) as avgSalary
  from MUTABAY.staffs staffs
)staffs
FULL OUTER JOIN MUTABAY.stores stores
ON staffs.store_id=stores.store_id
FULL OUTER JOIN MUTABAY.orders orders
ON stores.store_id = orders.store_id
FULL OUTER JOIN MUTABAY.order_items order_items
ON orders.order_id = order_items.order_id
FULL OUTER JOIN MUTABAY.products products
ON order_items.product_id = products.product_id
WHERE staffs.salary < staffs.avgsalary or order_items.discount > 0.05 or customer_id > 1500
GROUP BY store_name, first_name, salary, city, state, last_name, active
having (avg(staffs.salary) > 1000 OR state IS NOT NULL) OR (city = 'Aberdeen' AND active = 1)
ORDER BY store_name asc;
```

- **Stores Table => city column**

1. QUERY

```
SELECT first_name,last_name, active, salary, stores.store_name, stores.city, stores.state
FROM
(
  SELECT staffs.*,avg(salary) over (partition by store_id) as avgSalary
  from MUTABAY.staffs staffs
)staffs
FULL OUTER JOIN MUTABAY.stores stores
ON staffs.store_id=stores.store_id
FULL OUTER JOIN MUTABAY.orders orders
ON stores.store_id = orders.store_id
FULL OUTER JOIN MUTABAY.order_items order_items
ON orders.order_id = order_items.order_id
FULL OUTER JOIN MUTABAY.products products
ON order_items.product_id = products.product_id
WHERE staffs.salary < staffs.avgsalary or order_items.discount > 0.05 or customer_id > 1500
GROUP BY store_name, first_name, salary, city, state, last_name, active
having (avg(staffs.salary) > 1000 OR state IS NOT NULL) OR (city = 'Aberdeen' AND active = 1)
ORDER BY store_name asc;
```

- **Orders Table => order_status column**

5. QUERY

```
update MUTABAY.order_items set quantity_id =

  select quantity_id from MUTABAY.products products
  full outer join MUTABAY.order_items order_items on order_items.product_id = products.product_id
  full outer join MUTABAY.stocks stocks on stocks.product_id = products.product_id
  full outer join MUTABAY.orders orders on order_items.order_id=orders.order_id
  full outer join MUTABAY.customers customers on orders.customer_id=customers.customer_id
  full outer join MUTABAY.stores stores on orders.store_id=stores.store_id
  full outer join MUTABAY.staffs staffs on orders.staff_id=staffs.staff_id
  where products.product_id in
    (
      Select product_id from MUTABAY.order_items where order_id in
      (
        Select order_id from MUTABAY.orders
        WHERE
        (order_status = 1 AND ( shipped_date - required_date = 1 ))
        OR
        (order_status = 2 AND (shipped_date - required_date = 0 ))
      )

    )fetch next 1 rows only
);
```

6. Order_Items Table => discount column

6. QUERY

```
UPDATE MUTABAY.stores
SET MUTABAY.stores.store_name = (
  SELECT store_name FROM MUTABAY.stores stores
  INNER JOIN
  (
    SELECT order_i.staff_id, first_name, last_name, phone, email, order_i.store_id, manager_id, active, salary ,
    order_i.item_id ,order_i.product_id ,order_i.quantity_id ,order_i.discount ,order_i.customer_id ,order_i.order_status ,
    order_i.order_date ,order_i.required_date ,order_i.shipped_date
    FROM MUTABAY.staffs staffs
    FULL OUTER JOIN
    (
      SELECT orders.order_id, item_id, product_id, quantity_id, discount, orders.customer_id, orders.order_status,
          orders.order_date, orders.required_date, orders.shipped_date, orders.store_id, orders.staff_id
      FROM MUTABAY.order_items order_items
      FULL OUTER JOIN MUTABAY.orders orders
      ON orders.order_id = order_items.order_id
      WHERE ORDERS.ORDER_ID IN (SELECT ORDER_ID FROM MUTABAY.ORDER_ITEMS WHERE DISCOUNT > (SELECT
AVG(DISCOUNT) FROM MUTABAY.ORDER_ITEMS))
      OR
      (order_status = 2)
    ) order_i
    ON order_i.staff_id = staffs.staff_id
    WHERE (discount > 0.48 AND discount < 0.05) AND salary > 5000
    OR
    (active = 1 AND discount = 0.4)
  )order_i_staff
  ON order_i_staff.store_id = stores.store_id
  WHERE street='1 Fremont Point' or STATE IS NOT NULL
        fetch first 1 rows only );
```

IN MEMORY MANAGING COLUMN STORE

➔ IN MEMORY COLUMN STORE FOR LIST_PRICE AND MODEL_YEAR FROM PRODUCTS TABLE

```
CREATE TABLE MUTABAY.products_column(
  product_id INT PRIMARY KEY,
  product_name VARCHAR(100) NOT NULL,
```

```
    brand_id INT NOT NULL,
    category_id INT NOT NULL,
    model_year INT NOT NULL,
    list_price DECIMAL (10, 2) NOT NULL,
    FOREIGN KEY(category_id) REFERENCES MUTABAY.categories(category_id) ON DELETE CASCADE,
    FOREIGN KEY(brand_id) REFERENCES MUTABAY.brands(brand_id) ON DELETE CASCADE
) INMEMORY
INMEMORY MEMCOMPRESS FOR QUERY HIGH (list_price)
INMEMORY MEMCOMPRESS FOR CAPACITY HIGH (model_year)
NO INMEMORY(product_id, product_name, brand_id, category_id);
```

➔ IN MEMORY COLUMN STORE FOR CITY FROM STORES TABLE

```
CREATE TABLE MUTABAY.stores(
    store_id INT PRIMARY KEY,
    store_name VARCHAR(255) NOT NULL,
    phone VARCHAR(100),
    email VARCHAR(100),
    street VARCHAR(255),
    city VARCHAR(255),
    state VARCHAR(255),
    zip_code VARCHAR(50)
)INMEMORY
INMEMORY MEMCOMPRESS FOR QUERY HIGH (city)
NO INMEMORY(store_id, store_name, phone, email, street, state, zip_code);
```

➔ IN MEMORY COLUMN STORE FOR SALARY FROM STAFFS TABLE

```
CREATE TABLE MUTABAY.staffs(
    staff_id INT PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
```

```
    phone VARCHAR(100),
    email VARCHAR(100) NOT NULL,
    store_id INT NOT NULL,
    manager_id INT,
    active NUMBER(3,0) NOT NULL,
    salary NUMBER,
    FOREIGN KEY(store_id) REFERENCES MUTABAY.stores(store_id) ON DELETE CASCADE ,
    FOREIGN KEY(manager_id) REFERENCES MUTABAY.staffs(staff_id)
)INMEMORY
INMEMORY MEMCOMPRESS FOR CAPACITY HIGH (salary)
NO INMEMORY(staff_id, first_name, last_name, phone, email, store_id, manager_id, active);
```

➔ IN MEMORY COLUMN STORE FOR DISCOUNT FROM ORDER_ITEMS TABLE

```
CREATE TABLE MUTABAY.order_items(
    order_id INT,
    item_id INT,
    product_id INT NOT NULL,
    quantity_id INT NOT NULL,
    list_price DECIMAL (10, 2) NOT NULL,
    discount DECIMAL (10, 2) NOT NULL ,
    PRIMARY KEY(order_id, item_id),
    FOREIGN KEY(order_id) REFERENCES MUTABAY.orders(order_id) ON DELETE CASCADE ,
    FOREIGN KEY(product_id) REFERENCES MUTABAY.products(product_id) ON DELETE CASCADE
)INMEMORY
INMEMORY MEMCOMPRESS FOR CAPACITY HIGH (discount)
NO INMEMORY(order_id, item_id, product_id, quantity_id, list_price);
```

➔ IN MEMORY COLUMN STORE FOR ORDER_STATUS FROM ORDERS TABLE

```
CREATE TABLE MUTABAY.orders(
    order_id INT PRIMARY KEY,
    customer_id INT,
```

```
    order_status NUMBER(3,0),
        -- Order status: 1 = Pending; 2 = Processing; 3 = Rejected; 4 = Completed
    order_date VARCHAR2(20) NOT NULL,
    required_date VARCHAR2(20) NOT NULL,
    shipped_date VARCHAR2(20),
    store_id INT NOT NULL,
    staff_id INT NOT NULL,
    FOREIGN KEY(customer_id) REFERENCES MUTABAY.customers(customer_id) ON DELETE CASCADE ,
    FOREIGN KEY(store_id) REFERENCES MUTABAY.stores(store_id) ON DELETE CASCADE ,
    FOREIGN KEY(staff_id) REFERENCES MUTABAY.staffs(staff_id)
)INMEMORY
INMEMORY MEMCOMPRESS FOR QUERY HIGH(order_status)
NO INMEMORY(order_id, customer_id, order_date, required_date, shipped_date, store_id, staff_id);
```

➔ **JOIN GROUPS**

7. CREATE INMEMORY JOIN GROUP order_id_join (orders(order_id), order_items(order_id));
8. CREATE INMEMORY JOIN GROUP store_id_join (orders(store_id), stores(store_id));
9. CREATE INMEMORY JOIN GROUP staff_id_join (staffs(staff_id), orders(staff_id));
10. CREATE INMEMORY JOIN GROUP product_id_join (products(product_id), order_items(product_id));
11. CREATE INMEMORY JOIN GROUP customer_id_join (orders(customer_id), customers(customers_id));

Declare the scope of comparisons between compression methods.

Database In-Memory supports 5 levels of compression and we can even disable compression altogether if we want. As part of the ALTER TABLE/CREATE TABLE commands and the INMEMORY sub-clause, the options are:

- NO MEMCOMPRESS
  In-Memory data is populated without compression.
- MEMCOMPRESS FOR DML
  Level mainly intended for increasing DML performance and minimal compression.
- MEMCOMPRESS FOR QUERY LOW – the default
  Optimized for query performance (default).
- MEMCOMPRESS FOR QUERY HIGH
  Optimized for query performance and space saving
- MEMCOMPRESS FOR CAPACITY LOW
  Higher space saving level compared to Query High and Low
- MEMCOMPRESS FOR CAPACITY HIGH
  Level optimized for space saving and slightly less capacity.

The compression ratio can range from 2X to 20X, depending on the selected compression option, data type, and table contents. Each successive level typically decreases the amount of memory required to populate the object at a possible reduction in scan performance.