Análisis comparativo de bases de datos columnares para Data Analytics. Un caso práctico

Axel Nicolás Furlan Figueroa
Universidad Tecnológica Nacional, Facultad Regional Buenos Aires

Abstract
En el presente trabajo buscaremos medir el beneficio en performance provocado por la implementación del almacenamiento basado en columnas para consultas de tipo analíticas. Este tipo de consultas poseen problemas de performance en ambientes de grandes volúmenes de datos cuando se utilizan bases de datos transaccionales. Para realizar la medición, compararemos una base de datos columnar con una base de datos relacional. Utilizaremos diferentes consultas SQL correspondientes a diferentes ámbitos.

Palabras Clave
Big Data, Base de datos, Postgres, Redshift, OLAP, OLTP, Data Warehouse, Data Analytics

Introducción

En la última década ha surgido un gran interés por parte de las organizaciones en explotar grandes volúmenes de datos [1]. Surgen así los términos Big Data, Data Warehouse, Data Analytics, minería de datos y Data Lake.

En el presente trabajo, utilizaremos el término Big Data, para referirnos a grandes volúmenes de datos que, al querer ser procesados o leídos, presentan una complejidad considerable como para poder aplicar las tecnologías tradicionales vistas en sistemas de tipo OLTP [2].

Definimos OLTP como "procesamiento de transacciones en línea" y nos referiremos al uso de las bases de datos aplicadas a este tipo de sistemas. El acceso de este tipo de base está optimizado tanto para escritura como para lectura. Sin embargo, no todo tipo de lectura es performante.

El problema de utilizar bases OLTP para consultas analíticas es la baja performance, ya que no fueron diseñadas para tal finalidad.

Nuestro objetivo es medir los beneficios en performance de la implementación de la arquitectura de almacenamiento basado en columnas – o llamado *columnar.*

Realizaremos nuestra investigación utilizando dos ejemplos de bases de datos con dos arquitecturas diferentes: la columnar y la basada en filas. Nos enfocaremos en el uso de estas bases de datos para consultas –*queries*– que son comunes bajo un ambiente de análisis de Big Data y otras comunes en sistemas de OLTP. A su vez, también mediremos performance de escritura para entender, en profundidad, la naturaleza de la arquitectura de almacenamiento que ambas bases poseen.

Para este análisis, usaremos en concreto la base de datos open-source *Postgres* versión 8, y utilizaremos la versión presente en el día de la fecha de la base de datos columnar *Amazon Redshift*.

Breve historia de las bases de datos columnares

La primera base de datos columnar fue desarrollada en 1969. Llamada TAXIR, la misma fue elaborada para almacenar datos relacionados a la biología.

Cuando OLAP y el análisis y minado de datos cobraron importancia en las organizaciones, surgieron empresas que comenzaron a comercializar sus propias tecnologías de almacenamiento basado en columnas. He aquí que comenzaron a aparecer empresas como *ParAccel, Teradata y Vertica*.

A su vez, con la adopción masiva de las nubes públicas como *Amazon Web Services, Microsoft Azure* y *Google Cloud Platform*, cada una de ellas desarrolló una solución para Data Warehousing de forma "administrada" usando este tipo de bases de datos. Entre ellas están *Amazon Redshift, Google Bigquery* y *Azure Synapse*.

OLTP *vs.* OLAP

Con bases de datos de tipo OLTP, el servicio software es el que usa mayormente este recurso. La aplicación realiza *queries* para, por ejemplo, validar el email y contraseña de un usuario que está intentando acceder al sistema; buscar y retornar las transacciones del usuario para que el sistema las muestre en pantalla y; retornar el catálogo de productos disponibles en una aplicación de e-commerce.

A su vez, el sistema constantemente escribe datos en este tipo de BD, por ejemplo: nuevos usuarios, nuevas transacciones, cambios de estado de órdenes de compra, adiciones de productos al carrito de compras, etc.

El propósito de las bases de datos OLAP no es transaccional, sino *analítico* como su acrónimo describe. En este caso, los analistas de datos buscan *información* – datos que cuentan una historia. En este tipo de almacén de datos, también llamado *Data Warehouse*, las escrituras son poco frecuentes (comúnmente 1 vez al día) y se usa un paradigma de *bulk-update* para traer los nuevos datos a este almacenamiento.

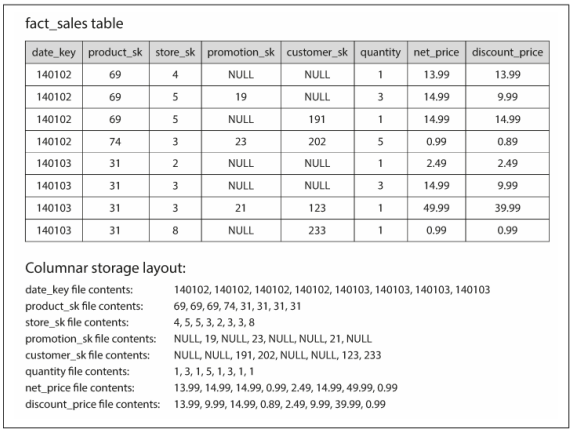
Este nuevo "propósito" para un almacén de datos permitió a los ingenieros probar diferentes enfoques para beneficiar la performance de tiempos de lectura, pero al mismo tiempo aumentando el tiempo de escritura – recordemos que esto no es problema para bases de datos donde las escrituras son poco frecuentes.

El enfoque columnar

Cuando se comienza a estudiar las bases de datos, comúnmente se hacen analogías con software de hojas de cálculo como Excel o con documentos de tipo *csv – comma–separated values*.

En los mismos, se pueden visualizar registros, uno debajo de otro, en un formato de "tabla" (no confundir con una *tabla* de una base de datos SQL). Este es el tipo de almacenamiento usado en las bases de datos relacionadas a sistemas de tipo OLTP.

En cambio, las bases de datos columnares almacenan la información de forma *pivotada*, donde cada línea en la página es una columna. A su vez, cada columna posee su archivo.


fact_sales table

date_key	product_sk	store_sk	promotion_sk	customer_sk	quantity	net_price	discount_price
140102	69	4	NULL	NULL	1	13.99	13.99
140102	69	5	19	NULL	3	14.99	9.99
140102	69	5	NULL	191	1	14.99	14.99
140102	74	3	23	202	5	0.99	0.89
140103	31	2	NULL	NULL	1	2.49	2.49
140103	31	3	NULL	NULL	3	14.99	9.99
140103	31	3	21	123	1	49.99	39.99
140103	31	8	NULL	233	1	0.99	0.99

Columnar storage layout:

date_key file contents: 140102, 140102, 140102, 140102, 140103, 140103, 140103, 140103
product_sk file contents: 69, 69, 69, 74, 31, 31, 31, 31
store_sk file contents: 4, 5, 5, 3, 2, 3, 3, 8
promotion_sk file contents: NULL, 19, NULL, 23, NULL, NULL, 21, NULL
customer_sk file contents: NULL, NULL, 191, 202, NULL, NULL, 123, 233
quantity file contents: 1, 3, 1, 5, 1, 3, 1, 1
net_price file contents: 13.99, 14.99, 14.99, 0.99, 2.49, 14.99, 49.99, 0.99
discount_price file contents: 13.99, 9.99, 14.99, 0.89, 2.49, 9.99, 39.99, 0.99

Figura 1 - almacenamiento basado en filas y almacenamiento basado en columnas [3]

El almacenamiento basado en columnas se basa en que cada columna posee un archivo propio y donde el orden de cada elemento en un archivo se condice con el orden del resto de los archivos. De esta forma, si el motor tuviese que ensamblar la fila número 23, tendría que recorrer todos los archivos de cada columna y juntar el registro número 23 de cada columna, y así formar la fila completa.

Análisis práctico - Benchmarking

Para el análisis de performance, utilizaremos un *dataset* público de la plataforma *Kaggle* correspondiente a accidentes de tránsito ocurridos en Estados Unidos desde Febrero 2016 hasta Diciembre 2020. El dataset pesa alrededor de 1 GB y posee 2.906.610 filas y 47 columnas [1].

El mismo dataset será cargado en una base de datos Postgres versión 8 y en Amazon Redshift.

Ambas bases de datos estarán alojadas en la nube de *Amazon Web Services* utilizando prestaciones similares de procesamiento y almacenamiento (16GB de RAM y 4 vCPU). Este tipo de instancias poseen el nombre de *large* en la terminología de AWS.

La justificación de la elección de bases de datos, y de la nube en particular, tienen que ver con el ambiente de pruebas. El objetivo es llevar a cabo las pruebas con la menor diferencia de prestaciones posibles.

A su vez, Amazon informa que la tecnología Redshift está basada en PostgreSQL[2].

Para el benchmarking, ejecutaremos las siguiente queries sobre la tabla de pruebas en ambas bases de datos:

```
SELECT COUNT(id)
FROM TEST_TABLE;

SELECT MAX(Severity)
FROM TEST_TABLE;

SELECT MIN(Severity)
FROM TEST_TABLE;

SELECT AVG(Severity)
FROM TEST_TABLE;

SELECT SUM(Severity)
FROM TEST_TABLE;

SELECT *
FROM TEST_TABLE
LIMIT 500;

SELECT Street
FROM TEST_TABLE
LIMIT 500;

SELECT COUNT(id)
FROM TEST_TABLE
WHERE id = 'A-5000';

SELECT Street
FROM TEST_TABLE
WHERE id = 'A-5000';

INSERT INTO TEST_TABLE
(<columnas>) VALUES
(<valores>);

UPDATE TEST_TABLE SET Street =
'calle 1234'
WHERE id = 'A-5000';

DELETE FROM TEST_TABLE
WHERE id = 'A-5000';
```

El objetivo de estas consultas es medir la performance – específicamente el *throughput* – de las operaciones básicas de SQL: SELECT, INSERT, UPDATE, DELETE.

[1] https://www.kaggle.com/sobhanmoosavi/us-accidents

[2] https://docs.aws.amazon.com/redshift/latest/dg/c_redshift-and-postgres-sql.html

Hipótesis del benchmark

Habiendo estudiado las diferencias entre el almacenamiento y lógica de consulta en bases de datos columnares y las basadas en filas, nuestra hipótesis es que la base de datos columnar tendrá una excelente performance en las consultas en las que se realizan *agregaciones* sobre una columna en particular. Con esto nos referimos a operaciones de SUM, COUNT, AVG, MAX y MIN.

Sin embargo, pensamos que tendrá baja performance comparado con la base de datos basada en filas en aquellas operaciones de escritura y de consulta de datos crudos, sin agregación.

La justificación de nuestra hipótesis se debe a lo expuesto en la sección "El enfoque columnar". En el caso de consultas con una agregación sobre una columna, la base de datos columnar sólo tiene que acceder a un archivo que contiene únicamente los datos correspondientes a esa columna, esto significa que el motor de consulta sólo debería realizar la lectura de este archivo (que puede estar distribuido y particionado en nodos) y realizar la función agregada sobre estos datos.

Esto contrasta con el enfoque en filas, ya que el motor de consulta tendrá que acceder a la fila completa compuesta por las 47 columnas de la tabla, para ir almacenando en memoria el valor de la columna que queremos consultar. Es decir, que aunque nosotros queramos acceder a una columna, inevitablemente el motor tendrá que leer la fila completa, para poder pasar a la siguiente y así sucesivamente.

Si nos referimos a una consulta cuyo objetivo es retornar los datos crudos de una tabla, el enfoque columnar debería presentar una performance reducida comparada a la misma del enfoque en filas. Esto sucede porque la BD columnar tendrá que acceder a múltiples archivos (1 por cada columna), guardar sus valores en memoria y realizar una operación de *merge* para entrelazar cada valor de cada columna en filas, y así poder retornar el resultado final. Mientras que en una base de datos OLTP, se tendrá que realizar lecturas de las páginas [4] que forman la tabla y retornar lo leído sin realizar ninguna operación especial para "acomodar" los valores.

Resultados de las pruebas

Cada consulta fue ejecutada unas 10 veces en ambas bases de datos.

Las pruebas de INSERT y DELETE fueron hechas de forma encadenada, es decir, una fila era insertada primero con el mismo *id* que la instrucción de DELETE usaría para luego eliminarla.

A continuación, enunciamos los resultados de las pruebas de cada consulta, medidas en milisegundos, en ambas bases de datos.

Query	Postgres	Redshift
`SELECT COUNT(id) FROM TEST_TABLE;`	940 ms 861 ms 779 ms 856 ms 763 ms 817 ms 868 ms 766 ms 828 ms 860 ms	694 ms 789 ms 641 ms 600 ms 743 ms 826 ms 898 ms 480 ms 460 ms 538 ms
`SELECT MAX(Severity) FROM TEST_TABLE;`	827 ms 861 ms 810 ms 790 ms 847 ms 826 ms 764 ms 882 ms 765 ms 770 ms	952 ms 451 ms 830 ms 809 ms 542 ms 578 ms 669 ms 618 ms 513 ms 399 ms
`SELECT`	768 ms	735 ms

`MIN(Severity) FROM TEST_TABLE;`	776 ms 829 ms 784 ms 822 ms 776 ms 824 ms 768 ms 772 ms 773 ms	759 ms 733 ms 540 ms 406 ms 561 ms 543 ms 470 ms 414 ms 617 ms
`SELECT AVG(Severity) FROM TEST_TABLE;`	810 ms 774 ms 845 ms 914 ms 884 ms 836 ms 885 ms 850 ms 845 ms 863 ms	482 ms 486 ms 647 ms 448 ms 674 ms 555 ms 550 ms 499 ms 406 ms 489 ms
`SELECT SUM(Severity) FROM TEST_TABLE;`	771 ms 759 ms 779 ms 789 ms 791 ms 857 ms 876 ms 773 ms 835 ms 854 ms	636 ms 651 ms 527 ms 553 ms 497 ms 432 ms 556 ms 448 ms 449 ms 526 ms
`SELECT * FROM TEST_TABLE LIMIT 500;`	944 ms 387 ms 378 ms 369 ms 375 ms 368 ms 379 ms 371 ms 366 ms 954 ms	969 ms 958 ms 883 ms 713 ms 639 ms 704 ms 874 ms 834 ms 694 ms 801 ms
`SELECT Street FROM TEST_TABLE LIMIT 500;`	195 ms 186 ms 187 ms 193 ms 192 ms 187 ms 253 ms	757 ms 598 ms 722 ms 901 ms 1008 ms 917 ms 627 ms
	190 ms 187 ms 191 ms	1236 ms 871 ms 665 ms
`SELECT COUNT(id) FROM TEST_TABLE WHERE id = 'A-5000';`	810 ms 727 ms 778 ms 745 ms 733 ms 691 ms 738 ms 713 ms 835 ms 705 ms	702 ms 586 ms 579 ms 797 ms 482 ms 385 ms 455 ms 423 ms 389 ms 424 ms
`SELECT Street FROM TEST_TABLE WHERE id = 'A-5000';`	707 ms 743 ms 693 ms 729 ms 759 ms 795 ms 696 ms 716 ms 747 ms 717 ms	862 ms 719 ms 654 ms 848 ms 804 ms 608 ms 867 ms 93 ms 786 ms 994 ms
`INSERT INTO TEST_TABLE (<columnas>) VALUES (<valores>);`	200 ms 189 ms 194 ms 187 ms 190 ms 193 ms 194 ms 188 ms 190 ms 198 ms	1841 ms 1364 ms 1025 ms 1306 ms 3057 ms 1327 ms 1747 ms 1091 ms 781 ms 972 ms
`UPDATE TEST_TABLE SET Street = 'calle 1234' WHERE id = 'A-5000';`	712 ms 690 ms 769 ms 757 ms 716 ms 693 ms 725 ms 707 ms 696 ms 711 ms	3995 ms 3245 ms 3280 ms 3321 ms 3313 ms 3403 ms 3201 ms 3396 ms 3561 ms 3018 ms
`DELETE FROM TEST_TABLE`	723 ms 687 ms	1563 ms 1123 ms

`WHERE id = 'A-5000';`	803 ms 711 ms 711 ms 751 ms 705 ms 740 ms 740 ms 742 ms	1728 ms 1260 ms 2307 ms 915 ms 892 ms 911 ms 855 ms 1009 ms

Análisis estadístico de los resultados

Para el siguiente análisis, agrupamos los resultados de las consultas de funciones agregadas en una sola muestra. También agrupamos los resultados de aquellas consultas de datos crudos que poseen un filtro de WHERE con las que no lo usan.

Queremos determinar con un nivel de significancia $\alpha = 0,05$ si podemos aceptar las hipótesis alternativas listadas a continuación (siendo las hipótesis nulas, la contraparte de estas):

1. Los tiempos de las consultas agregadas son menores en la base de datos columnar.
2. Los tiempos de las consultas de datos crudos (SELECT *) son mayores en la base de datos columnar.
3. Los tiempos de selección de una columna específica son menores en la base de datos columnar.
4. Los tiempos de INSERT son mayores en la base de datos columnar.
5. Los tiempos de UPDATE son mayores en la base de datos columnar.
6. Los tiempos de DELETE son mayores en la base de datos columnar.

Para este análisis, utilizaremos la prueba *T de* Student (también conocida como *T-Test*).

Para la siguiente prueba, asumimos que cumplimos los 3 requerimientos para realizar una prueba de T de Student:

1. Las 2 muestras son independientes.
2. Ambas muestras siguen una distribución normal.
3. Las desviaciones estándar de ambas muestras son aproximadamente iguales.

Nos enfocaremos en la prueba independiente de una cola *(one-tailed)* de dos muestras de igual tamaño y de iguales varianzas.

La *t* se define por la siguiente ecuación:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{s_p \sqrt{\frac{2}{n}}}$$

Donde *Sp* se expresa como:

$$s_p = \sqrt{\frac{s_{X_1}^2 + s_{X_2}^2}{2}}.$$

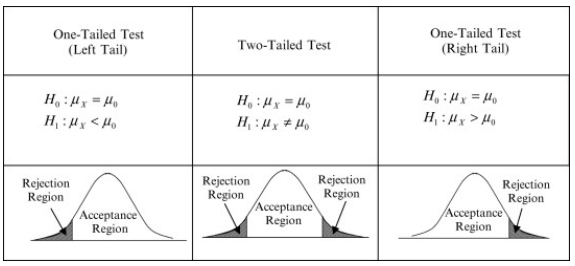


Figura 2 - tipos de T-Test

A continuación, listamos los resultados enumerados para cada una de las hipótesis mencionadas anteriormente.

Aclaración: la *población 1* corresponde a la muestra de la BD Postgres, la *población 2* corresponde a Redshift.

A su vez, la orientación del mayor y menor en las poblaciones indican la hipótesis nula. Si el *p-valor* obtenido es menor a nuestro nivel de significancia $\alpha = 0,05$podemos rechazar esa hipótesis nula y aceptar nuestras hipótesis, enunciadas anteriormente.

Los resultados fueron:

1. Test Statistic t= 12.1565
   Population 1 < Population 2: P-Value = < .00001
2. Test Statistic t= -3.7414
   Population 1 > Population 2: P-Value = 0.0007
3. Test Statistic t= -3.9395
   Population 1 < Population 2: P-Value = 0.9998
4. Test Statistic t= -6.0875
   Population 1 > Population 2: P-Value = < .00001
5. Test Statistic t= -32.1315
   Population 1 > Population 2: P-Value = < .00001
6. Test Statistic t= -3.4946
   Population 1 > Population 2: P-Value = 0.0013

Conclusiones finales y trabajos futuros

Medimos la performance de una base de datos columnar y la comparamos con una relacional basada en filas. Para las pruebas no utilizamos índices, clusterización, paralelismo entre nodos ni claves de ordenación o distribución. Nos basamos púramente en el tipo de implementación del almacenamiento de datos.

Comprobamos lo ventajoso que es el almacenamiento basado en columnas para consultas que son comunes en sistemas OLAP.
Utilizamos una tabla con 2,9 millones de filas y 47 columnas para las pruebas. En consultas con funciones agregadas, vimos un incremento promedio del **29%** en rendimiento comparado con una base de datos relacional basada en filas Identificamos también un incremento de **38%** en una consulta para calcular el promedio de una columna. Si bien este incremento puede resultar en primer instancia poco determinante debemos tener en cuenta que estamos analizando el rendimiento de un sólo aspecto en particular de las bases de datos usadas para OLAP.

Una de las características de estas bases de datos es la facilidad para utilizar *clusterización*. Normalmente, las bases de datos columnares se usan en conjunto con una arquitectura distribuida de nodos para apalancarse del procesamiento de datos en paralelo, algo que no analizamos en el presente trabajo.

Adicionalmente, pudimos probar que las bases de datos columnares no son adecuadas para su uso en sistemas OLTP, ya que poseen tiempos de escritura y modificación de registros muy elevados comparados con bases de datos relacionales tradicionales. Nuestros resultados indican que para operaciones de escritura, en promedio, la penalidad en performance es de un **270%**.

A su vez, comprobamos 5 de las 6 hipótesis enunciadas. La hipótesis que no pudimos validar corresponde a la selección de una columna específica, que en la base de datos columnar resultó ser una operación significativamente más lenta comparada con la BD orientada a filas.

Esto puede tener sentido ya que este tipo de base de datos posee algoritmos de compresión [5] para reducir el espacio ocupado por una tabla, lo que podría implicar que el motor realiza varias lecturas para construir la columna con cada valor posible de la columna.
Puede ser interesante estudiar los beneficios en almacenamiento de ambos tipos de bases de datos en un trabajo futuro.

Referencias

[1] Chamakuzhi Subramanian, 2018. The Boom and Bust of Big Data Analytics. Consultado en Agosto de 2021. Link de consulta: https://www.researchgate.net/publication/329028850_The_Boom_and_Bust_of_Big_Data_Analytics
[2] What is Big Data?. Documentación oficial de Oracle. Consultado en Agosto de 2021. Link de

consulta:
https://www.oracle.com/big-data/what-is-big-data/
[3][5] Martin Kleppmann, 2017. Designing Data-Intensive Applications. Publicado por O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. Consultado en Agosto de 2021.
[4] Documentación oficial de PostgreSQL. Consultado en Agosto de 2021. Link de consulta: https://www.postgresql.org/docs/8.1/storage.html#STORAGE-FILE-LAYOUT