Práctica 3. NoSQL, Optimización y Transacciones

Versión 2.4 27/11/2020

Objetivos docentes

Tras la realización de la Práctica 3, el estudiante debe ser capaz de:

- Entender los conceptos involucrados en el uso y explotación de una base de datos NoSQL a través de MongoDB.
- Usar el **planificador** de PostgreSQL para analizar el coste de una consulta.
- Identificar posibles **índices** que pueden mejorar el rendimiento de una consulta.
- Identificar formas alternativas de realizar una consulta (reordenaciones y cambios en los 'join', uso de unión, 'except', …) para mejorar dicho rendimiento.
- Estudiar el impacto de la generación de **estadísticas**.
- Entender el funcionamiento de una **transacción**.
- Implementar transacciones con mecanismos de ROLLBACK.
- Entender la función de COMMIT.
- Entender los mecanismos de **bloqueo** y *deadlocks.*

1) NoSQL

En esta parte se trabajará con MongoDB, creando documentos específicos que serán consultados y mostrados desde Python en el proyecto de aplicación web desarrollado hasta el momento.

Se recuerda que, para la realización de las siguientes tareas, el servidor MongoDB debe estar activo en todo momento, y se debe usar la librería PyMongo para poder interactuar con MongoDB desde Python.

Tareas a Realizar

A. A partir de la Base de Datos de Películas para PostgreSQL, crear una Base de Datos documental que contenga una colección de documentos que representará las 800 películas estadounidenses más actuales en base a la información almacenada en la Base de Datos relacional. Los documentos tendrán la siguiente estructura:
   a. Título – cadena
   b. Géneros – listado de géneros
   c. Año – número
   d. Directores – listado de directores
   e. Actores participantes – listado de actores
   f. Hasta 10 películas más actuales y más relacionadas – listado de títulos y años
   g. Hasta 10 películas más actuales y relacionadas – listado de títulos y años

Las películas más relacionadas (f) serán aquellas que tengan una coincidencia en el 100% de los géneros, mientras que las relacionadas (g) serán aquellas que tengan una coincidencia en el 50%. Ambos conjuntos de películas relacionadas deben ser disjuntos y, obviamente, la película (a) no puede aparecer como película relacionada (en f o g). Si la película (a) tuviera un solo género, sólo se indicarán las películas más relacionadas (f).

Un ejemplo de documento sería el siguiente (utilizad, a la hora de crear la Base de Datos en MongoDB, estos mismos nombres de etiquetas):

```
{
    title: "Toy Story 2",
    genres: ["Adventure", "Animation", "Comedy", "Family", "Fantasy"],
    year: 1999,
    directors: ["Brannon, Ash", "Lasseter, John", "Unkrich, Lee"],
    actors: ["Benson, Jodi", "Burton, Corey", ...],
    most_related_movies: [{title:"Antz", year:1998}, {title:"Quest for Camelot", year:1998}],
    related_movies: [
        {title:" Adventures of Rocky & Bullwinkle, The", year:2000},
        {title:" Road to El Dorado, The", year:2000},
        {title:" Bug's Life, A", year:1998},
        {title:" Mulan", year:1998},
        {title:" Rugrats Movie, The", year:1998},
        {title:" Hercules, The", year:1997}
    ]
}
```

B. Para crear de manera automática la Base de Datos documental referida en el punto anterior, se programará un script en Python, que extraerá la información de la BD PostgreSQL (cuyo script de creación `dump_v1.2-P3.sql` podéis encontrar en Moodle para esta práctica) y se encargará de crear otra BD en MongoDB llamada `si1`, que contendrá la colección llamada `topUSA` donde se almacenarán los documentos. Este script deberá estar situado en la carpeta `app` del proyecto, en un fichero llamado **`createMongoDBFromPostgreSQLDB.py`**, y se ejecutará manualmente antes de arrancar la aplicación web para crear los datos necesarios sobre los que trabajar. Para extraer los datos de PostgreSQL se utilizará el framework SQLAlchemy.

C. La Base de Datos creada en MongoDB se consultará desde la aplicación web construida en prácticas anteriores, utilizando consultas y órdenes MongoDB. Para ello, se habilitará un nuevo enlace en el menú de navegación llamado "Top USA".

D. Al pulsar en el nuevo enlace referido anteriormente, se mostrará una página con 3 tablas distintas que contendrán la siguiente información, como resultado de las consultas realizadas a los documentos contenidos en `si1.topUSA` en MongoDB:
   a. Una tabla con toda la información de aquellas películas (documentos) que sean comedias del año 1997, y que contengan la palabra "Life" en el título.
   b. Una tabla con toda la información de aquellas películas (documentos) que hayan sido dirigidas por Woody Allen en los años 90.
   c. Una tabla con toda la información de aquellas películas (documentos) en las que Jim Parsons y Johnny Galecki hayan compartido reparto.

2) Optimización

Base de Datos de trabajo

Para la parte de Optimización y Transacciones (apartados 2 y 3, respectivamente) de esta práctica, se suministra una variante de la Base de Datos de la Práctica anterior (`dump_v1.2-P3.sql.zip`). En dicha Base de Datos se ha hecho uso de sentencias:

```
ALTER TABLE table_name SET (autovacuum_vacuum_threshold=100000000,
autovacuum_analyze_threshold=100000000);
```

para evitar la generación automática de estadísticas, y se ha realizado algo de limpieza sobre la BD de la práctica anterior, además de cargarse los importes de los pedidos.

Tareas a Realizar

E. Estudio del impacto de un **índice**:
   a. Crear una consulta, **clientesDistintos.sql**, que muestre el número de clientes distintos que tienen pedidos en un mes dado usando el formato YYYYMM (por ejemplo 201504), con importe (totalamount) superior a un umbral dado, por ejemplo 100.
   b. Mediante la sentencia EXPLAIN estudiar el plan de ejecución de la consulta.
   c. Identificar un índice que mejore el rendimiento de la consulta y crearlo (si ya existía, borrarlo).
   d. Estudiar el nuevo plan de ejecución y compararlo con el anterior.
   e. Probad distintos índices y discutid los resultados.
   f. Modificar el *script* **clientesDistintos.sql**, añadiendo las sentencias de creación de índices y de planificación.

   Ayuda:
   - Tened presente que un índice sobre la columna A y la columna B no es equivalente a un índice sobre la columna B y la columna A.
   - Por otro lado, se recuerda que no es lo mismo count(distinct … que distinc(count …
   - Si se crea un índice sobre una columna, pero luego en la consulta se usa una función sobre ella, el planificador no va a hacer uso del índice. Postgresql permite crear índices sobre funciones (más recomendable).

F. Estudio del impacto de cambiar la **forma** de realizar una **consulta**:
   a. Estudiar los planes de ejecución de las consultas alternativas mostradas en el Anexo 1 y compararlos.
      i. ¿Qué consulta devuelve algún resultado nada más comenzar su ejecución?
      ii. ¿Qué consulta se puede beneficiar de la ejecución en paralelo?

G. Estudio del impacto de la generación de **estadísticas**:
   a. Partir de la base de datos suministrada limpia (recién creada y cargada de datos).
   b. Estudiar con la sentencia EXPLAIN el coste de ejecución de las dos consultas indicadas en el Anexo 2.
   c. Crear un índice en la tabla *orders* por la columna *status*.
   d. Estudiar de nuevo la planificación de las mismas consultas.
   e. Ejecutar la sentencia ANALYZE para generar las estadísticas sobre la tabla *orders*.
   f. Estudiar de nuevo el coste de las consultas y comparar con el coste anterior a la generación de estadísticas. Comparar el plan de ejecución y discutir el resultado.
   g. Comparar con la planificación de las otras dos consultas proporcionadas y comentar los resultados.
   h. Crear un *script* **countStatus.sql**, conteniendo las consultas, la creación de índices y las sentencias ANALYZE.

   Ayuda:
   - ¿Qué hace el generador de estadísticas?
   - Usar la sentencia ANALYZE, no VACUUM ANALYZE
   - EXPLAIN ANALYZE no calcula estadísticas (¡y ejecuta la consulta!)). El cálculo de estadísticas se realiza con la sentencia ANALYZE
   - ¿Por qué la planificación de las dos consultas es la misma hasta que se generan las estadísticas?

3) Transacciones

Tareas a Realizar

H. Estudio de **transacciones**: A partir del esqueleto suministrado (`FicherosAuxiliaresP3.zip`), realizar una página, **borraCliente**, que ejecute una transacción que borre un cliente, y toda su información asociada (carrito e historial y pedidos con su detalle), de la BD.
   a. La página recibirá vía GET el *customerid* del cliente a borrar, que será solicitado mediante un formulario por la propia página.
   b. La transacción deberá tener mecanismos de *rollback*.
   c. Se usarán sentencias SQL (vía *execute()*) para gestionar la transacción. Opcionalmente se podrá además (por ejemplo controlado por un argumento pasado vía GET) hacer uso de la interfaz SQLAlchemy (*begin()*, etc.) de gestión de transacciones en lugar de *execute()*.
   d. Para poder realizar este ejercicio, se deberá desactivar (eliminar) en la base de datos, si las hubiera, todas las restricciones ON DELETE CASCADE referentes al cliente y sus pedidos, pero manteniendo las *foreign keys* que aseguran la integridad. De esta forma no se propagará el borrado de un cliente de forma automática a los registros asociados, y podremos hacer el borrado manual de dichos registros.
   e. Elaborar una versión de la página, controlada por un argumento, que implemente este borrado de registros en un orden incorrecto, de forma que se provoque un fallo de restricción de *foreign key*.
   f. Se deberá entonces realizar un *rollback* de la transacción para volver a la situación original, deshaciendo los cambios realizados hasta ese momento.
   g. Elaborar otra versión con el orden de borrado correcto, y que funcione de la forma esperada y borre todos los registros asociados a un cliente.
   h. Se deberá realizar un control de errores adecuado, y no considerar el NOT FOUND como un error.
   i. Se mostrarán en la página resultante trazas de los cambios parciales que se van realizando, y cómo estos cambios se deshacen al realizar un *rollback*.
   j. Alterar la versión incorrecta de la página para que se realice algún COMMIT intermedio (seguido de BEGIN, **¿por qué?**), antes de producirse el error, y comprobar que los cambios realizados antes del COMMIT persisten tras el ROLLBACK. Controlar si se hace *commit* o no mediante otro argumento pasado a la página vía GET.

I. Estudio de **bloqueos** y *deadlocks*:
   a. Partir de una base de datos limpia (recién creada y cargada de datos).
   b. Crear un *script* **updPromo.sql**, que creará una nueva columna, *promo*, en la tabla *customers*. Esta columna contendrá un descuento (en porcentaje) promocional.
   c. Añadir al *script* la creación de un *trigger* sobre la tabla *customers* de forma que al alterar la columna *promo* de un cliente, se le haga un descuento en los artículos de su cesta o carrito del porcentaje indicado en la columna *promo* sobre el precio de la tabla *products*.
   d. Modificar el *trigger* para que haga un *sleep* durante su ejecución (sentencia PERFORM *pg_sleep*(nn) …) en el momento adecuado.
   e. Insertar también un *sleep* en el momento adecuado en la versión correcta de la página que borra un cliente (eliminar el COMMIT intermedio si es necesario).
   f. Crear uno o varios carritos (*status* a NULL) mediante la sentencia UPDATE.
   g. Acceder a la página que borra un cliente con un pedido en curso (cesta o carrito) y, a la vez, realizar un *update* (en una sesión psql/pgadmin3/etc.) de la columna *promo* del mismo cliente.

h. Comprobar en otra sesión que, durante el *sleep* en la página de borrado o el contenido en el *trigger*, los datos alterados por la página o por el trigger no son visibles. Comentad el porqué.
i. Revisar mediante pgadmin3 los bloqueos mientras duren los *sleep*. Comentadlos.
j. Ajustar el punto en que se hacen los *sleep* para conseguir un *deadlock* y explicar por qué se produce.
k. Discutir cómo afrontar o evitar este tipo de problema.

Ayuda:

- Es conveniente desactivar el auto-commit de PostgreSQL.
- Por otro lado, PostgreSQL suele ejecutar un rollback automático si se produce un error. Es una buena práctica que el programador controle directamente esta situación, ejecutando un *rollback* en caso de error (sin confiar en que lo haga el motor del SGBD) y un commit en caso contrario.
- El estándar POSIX de SQL define 'START TRANSACTION', en vez de 'BEGIN'. Indagar las especificaciones de PostgreSQL y SQLAlchemy al respecto.

Entregables

Como **resultado** de la práctica se entregará:

- Código fuente (separar en 3 carpetas distintas):
  - Scripts solicitados en los 3 apartados.
  - Proyecto completo de la P2 (páginas HTML, Python, etc.), pero con la integración MongoDB exigida. Es decir, con la nueva página "Top USA" y la funcionalidad que la *renderice* a través de las consultas a la BD de MongoDB.
  - Esqueleto de la aplicación exigida para el apartado H.
- Memoria en la que se incluirá:
  - Descripción del material entregado.
  - Resultados de cada ejercicio solicitado y su discusión.
  - Los planes de ejecución solicitados en el apartado de Optimización.
  - Los resultados intermedios de la ejecución de las transacciones de borrado, en su versión correcta e incorrecta, describiendo y explicando el funcionamiento de una transacción.
  - El funcionamiento de COMMIT.
  - Los resultados (capturas de pantalla) en las que se comprueba el funcionamiento de una transacción, y su nivel de aislamiento, y una descripción y explicación del proceso.
  - Capturas de pantalla que muestren los bloqueos, y su explicación.
  - Descripción de los bloqueos conseguidos y explicación del *deadlock*.

Fechas

Esta práctica tendrá una única entrega:

- Entrega final: Todos los grupos deberán entregar la documentación y todos los ficheros correspondientes a la práctica hasta las 23:59 horas del día 18 de diciembre de 2020.

Evaluación

Se valorará especialmente la explicación y discusión de los resultados obtenidos, así como cualquier mejora o añadido adicional que aporte el estudiante.

Bibliografía

Información sobre MongoDB y PyMongo:
https://docs.mongodb.com/manual
https://pymongo.readthedocs.io

Información sobre el planificador de PostgresSQL:
http://www.postgresql.org/docs/9.5/interactive/sql-explain.html
http://www.postgresql.org/docs/9.5/interactive/using-explain.html
https://sites.google.com/site/robertmhaas/presentations: ver 'The PostgreSQL Query Planner'

Información sobre el generador de estadísticas:
http://www.postgresql.org/docs/9.5/interactive/sql-analyze.html
http://www.postgresql.org/docs/9.5/interactive/monitoring-stats.html

Información sobre transacciones:
http://www.postgresql.org/docs/9.5/interactive/sql-commands.html: ver BEGIN, COMMIT, ROLLBACK, etc.
http://www.postgresql.org/docs/9.5/interactive/mvcc.html

Información sobre mecanismos de bloqueo:
http://www.postgresql.org/docs/9.5/interactive/mvcc.html
http://www.postgresql.org/docs/9.3/interactive/monitoring-locks.html
http://www.postgresql.org/docs/9.5/interactive/view-pg-locks.html

Anexo 1

Ejemplo para el Ejercicio F, impacto en la forma de realizar una consulta

A continuación, se muestran tres formas alternativas de construir una consulta:

```
select customerid
from customers
where customerid not in (
  select customerid
  from orders
  where status='Paid'
);
```

```
select customerid
from (
  select customerid
  from customers
  union all
  select customerid
  from orders
  where status='Paid'
) as A
group by customerid
having count(*) =1;
```

```
select customerid
from customers
except
  select customerid
  from orders
  where status='Paid';
```

Estudiar, explicar y comparar la planificación de las tres consultas.

Anexo 2

Ejemplo para el Ejercicio G, impacto en la generación de estadísticas

A continuación, se muestran dos consultas muy parecidas:

```
select count(*)
from orders
where status is null;
```

```
select count(*)
from orders
where status ='Shipped';
```

Estudiar, explicar y comparar la planificación de las dos consultas

- sobre una base de datos limpia (recién creada y cargada de datos),
- tras crear un índice
- y tras generar las estadísticas con la sentencia ANALYZE.

Comparar también con la planificación de las siguientes consultas, una vez generadas las estadísticas:

```
select count(*)
from orders
where status ='Paid';
```

```
select count(*)
from orders
where status ='Processed';
```