Práctica 4: Optimización, Transacciones y Seguridad

Versión 1.0 2017/11/26

Objetivos

Tras la realización de esta práctica el alumno debe ser capaz de:

- 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 preparar sentencias SQL.
- 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.*
- Comprender cómo acceder a información sensible de una Base de Datos a través de su catálogo.
- Entender las técnicas de acceso indebido a sitios web, aprovechando vulnerabilidades, mediante inyección de código SQL, y cómo obtener información contenida en la BD.

Enunciado: Optimización

A. 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, por ejemplo 201404, 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.

B. Estudio del impacto de preparar sentencias SQL:
   a. A partir del esqueleto suministrado, crear una página PHP, **listaClientesMes.php**, que use la consulta anterior para mostrar el número de clientes distintos que tienen pedidos en un mes dado con importe (*totalamount*) superior a un umbral dado.
   b. La página pedirá el mes a mostrar y entonces construirá una tabla con el número de clientes distintos para umbrales equiespaciados (por ejemplo: 0, 5, 10,...) hasta llegar al máximo umbral (el número de clientes es cero). Esto se realizará invocando a la consulta anterior dentro de un bucle PHP.
   c. Modificar la página para que se haga un PREPARE de la consulta y comparar el rendimiento.

d. Discutid los resultados, con y sin el índice del apartado anterior, y tras generar estadísticas (ver más abajo).

   Ayuda: Se suministra esqueleto de la página PHP.
   Ejecutad varias veces la misma consulta para evitar el efecto de la *caché* de disco.

C. Estudio del impacto de cambiar la forma de realizar una consulta:
   a. Estudiar los planes de ejecución de las consultas alternativas mostradas en el Apéndice 1 y compararlos.

D. Estudio del impacto de la generación de estadísticas:
   a. Partir de la base de datos suministrada limpia.
   b. Estudiar con la sentencia EXPLAIN el coste de ejecución de las dos consultas indicadas en el Apéndice 1.
   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 consulta 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**, con las consultas, creación de índices y sentencias ANALYZE.

   Ayuda:
   - ¿Qué hace el generador de estadísticas?
   - Usar la sentencia ANALYZE, no VACUUM ANALYZE
   - EXPLAIN ANALYZE no calcula estadísticas. El cálculo de estadísticas se realiza con la sentencia ANALYZE

Enunciado: Transacciones y *deadlocks*

E. Realizar una página PHP, **borraCliente.php**, 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 *exec()*) para gestionar la transacción.
   Opcionalmente se podrá además (por ejemplo controlado por un argumento pasado vía GET) hacer uso de la interfaz PDO (*beginTransaction()*, etc.) de gestión de transacciones en lugar de *exec()*.
   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 página **borraClienteMal.php** 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 página **borraCliente.php** con el orden de borrado correcto, y que funcione de la forma esperada y borre todos los registros asociados a una 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.

F. Estudio de bloqueos y *deadlocks*:
a. Partir de una base de datos limpia.
b. Crear un *script*, **updPromo.sql**, que cree 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 página PHP que borra un cliente (eliminar el COMMIT intermedio).
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/phppgadmin/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 PHP o por el trigger no son visibles. Comentad el porqué.
i. Revisar mediante phppgadmin 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.

Enunciado: Seguridad

G) Acceso indebido a un sitio web

El objetivo de este ejercicio es acceder a nuestro sitio web, protegido con usuario y contraseña, sin disponer ni de usuario ni de contraseña.

Se suministra una página ejemplo de validación de login (**xLoginInjection.php**). Revisar el código fuente y detectar la vulnerabilidad.

Se pide:
a) Conociendo el nombre de login de un usuario (pero no su contraseña), conseguir validar el proceso de login.
Como usuario se puede usar cualquiera de la BD, por ejemplo, *gatsby*.
Para conseguir validar el login, se deberá inyectar código SQL en el campo de contraseña.
b) Usar la misma técnica para validar el login sin conocer ni el login del usuario ni su contraseña.
c) Discutir cómo evitar estos accesos indebidos.

H) Acceso indebido a información

El objetivo de este ejercicio es extraer la lista de clientes de nuestro sitio web.

Se suministra una página (**xSearchInjection.php**) que nos permitirá acceder a estos datos. Esta página muestra los títulos de las películas de un año determinado, suministrado por el usuario, mediante una consulta SQL aparentemente correcta.

Asumiremos que, de algún modo, hemos conseguido averiguar (o simplemente sospechamos) que el sitio web hace uso de una base de datos sobre PostgreSQL, pero, obviamente, no sabemos nada más acerca de la estructura interna de la base de datos.

Haremos uso de las tablas de catálogo de PostgreSQL. La bibliografía contiene un enlace a la información sobre dicho catálogo.

En el ejercicio, el alumno debe tomar el papel de un atacante al sitio web (al que ha podido acceder), así que no debe tener conocimiento del código fuente de la página PHP, pero sí del código HTML que se ha descargado en su navegador.

Se pide:

a) Se sospecha que la consulta que se realiza en la página es del tipo:
`select` *`columna_titulo`* `from` *`tabla_peliculas`* `where` *`columna_anio`* `= '$variable'`,
pero se desconocen los nombres de las tablas y columnas.
(En realidad es algo más compleja, pero ello no afecta al proceso de inyección: ¿por qué?)
b) Encontrar una cadena de búsqueda que nos muestre todas las tablas del sistema.
Para ello hacer uso de la tabla *pg_class*, descrita en la documentación de PostgresSQL.
c) Encontrar una cadena de búsqueda que sólo muestre las tablas de interés de la base de datos (schema public), y no las internas de PostgresSQL.
Para ello encontrar el 'oid' del schema 'public' en la tabla pg_namespace y luego filtrar el resultado del apartado anterior con este oid.
Nota: La columna 'oid' es interna, y no se muestra en un select * …, pero sí cuando se pide explícitamente con select oid,* …
d) Identificar en la lista anterior la tabla candidata a contener información de los clientes.
e) Encontrar una cadena de búsqueda que nos muestre el 'oid' de esta tabla consultando la tabla pg_class.
f) Mediante el 'oid' anterior, encontrar una cadena de búsqueda que nos muestre las columnas de la tabla candidata, consultando la tabla pg_attribute.
g) Identificar la columna candidata a contener los clientes del sitio web.
h) Con la información anterior, encontrar una cadena de búsqueda que nos muestre la lista de clientes.
i) ¿Se resolvería el problema usando una lista desplegable (*combobox*) en lugar de un campo de texto para introducir el año? ¿Y usando el método POST en lugar de GET en la página PHP?

   Ayuda: ¿Cómo se juntan los resultados de varias consultas?

Resultados a entregar

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

- Código fuente:
  - *Scripts* de las consultas.
  - Páginas PHP.

- 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.
  - Su descripción y análisis, así como el impacto de usar PREPARE.
  - 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*.
  - Los valores de los campos *login* y contraseña que permiten el acceso indebido al sitio web, describiendo el proceso.
  - Discutir qué técnicas usar para evitar estos ataques (consultar la bibliografía).
  - El motivo por el que la página suministrada permite el acceso indebido a la información de la BD.
  - Las cadenas de texto solicitadas que nos permiten el acceso indebido a la información, describiendo qué consultas realizan y de dónde obtienen la información.
  - Capturas de los resultados de dichas cadenas de texto.
  - Respuesta a la pregunta planteada sobre el uso de una *combobox* y el método POST.
  - Medidas sugeridas para proteger el acceso a la información

Base de Datos de trabajo

En esta práctica se suministra una variante de la Base de Datos suministrada en la Práctica anterior con la que se realizarán las pruebas de lo solicitado en la práctica.

En dicha BD 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.

Entrega

Esta práctica se dividirá en tres entregas, una cada semana. Las dos primeras entregas, si bien obligatorias, no serán evaluadas y servirán para ver el progreso del proyecto. El contenido de las entregas será:

1. Primera semana: Apartado de Optimización
2. Segunda semana: Apartado de Transacciones.
3. Entrega final: Se deberán entregar todos los ficheros correspondientes y documentación relativos a la práctica.

Evaluación

Se valorará especialmente la explicación y discusión de los resultados obtenidos.

Se valorará positivamente cualquier mejora o añadido adicional que aporte el alumno.

Bibliografía

La información sobre el planificador de PostgresSQL se puede consultar en:
http://www.postgresql.org/docs/9.3/interactive/sql-explain.html
http://www.postgresql.org/docs/9.3/interactive/using-explain.html
https://sites.google.com/site/robertmhaas/presentations

sobre el generador de estadísticas en:
http://www.postgresql.org/docs/9.3/interactive/sql-analyze.html
http://www.postgresql.org/docs/9.3/interactive/monitoring-stats.html

sobre transacciones en:
http://www.postgresql.org/docs/9.3/interactive/sql-commands.html: ver BEGIN, COMMIT, ROLLBACK, etc.
http://www.postgresql.org/docs/9.3/interactive/mvcc.html
http://es.php.net/manual/en/book.pdo.php

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

sobre el catálogo en PostgresSQL en:
http://www.postgresql.org/docs/9.3/static/catalogs.html

y sobre vulnerabilidades en aplicaciones en:
https://www.owasp.org/index.php/Main_Page

Apéndice 1

Ejemplo para el Ejercicio C, impacto de 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.

Ejemplo para el Ejercicio D, impacto de 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';
```