Trabajo Práctico: Cálculo de reputación de un sitio de e-commerce

72.41 - Bases de Datos II

Integrantes:

Chidiac, Kevin

Lobo, Daniel

Latron, Romain

Instituto Tecnológico de Buenos Aires

1er. Cuatrimestre 2018

Lunes 5 de Junio

Índice

Objetivos

Realizar modificaciones para mejorar la performance de un script Oracle SQL de una base de datos de una empresa de e-commerce, puntualmente sobre el proceso de actualización de la reputación de vendedores.

Para solucionar esto se realizaron mejoras de dos tipos:

1. Mejoras sobre el esquema
2. Mejoras sobre los procedures

El script que hemos mejorado es el *calculate_seller_scoring.sql*, el mismo actualiza los puntajes de los vendedores a partir de los reviews que les otorgan los clientes/usuarios.

En el contenido de este informe se explican:

1. Los scripts de ejecución de las modificaciones realizadas y propuestas.
2. La justificación teórica de las mismas.
3. Resultados experimentales de las mejoras.

Lógica de negocio y explicación del proceso

Se trata de una empresa de e-commerce donde se registran las ventas de artículos. Cada venta es registrada en la tabla de SALE. Y por cada venta se generan dos reviews, una review que hace el vendedor y otra review que hace el comprador.

Al mismo tiempo cada review tiene las siguientes características:

1. Calidad: POSITIVA, NEUTRA, NEGATIVA
2. Estado: PUBLISHED, PENDING
3. Fechas: CREATED, MODIFIED

El proceso calcula la reputación de cada vendedor y se ejecuta toda las noches. Los cambios se ven reflejados en la tabla SCORE. En dicha tabla se suman las calificaciones, POSITIVA, NEUTRA, NEGATIVA. Al mismo tiempo, las calificaciones se agrupan de manera semanal, mensual y cada 6 meses.

De esta manera se calcula la “reputación” de cada vendedor.

Archivos

Los archivos utilizados en este proyecto se pueden ver en la raíz del mismo y se listan a continuación:

- entregable.sql
- Informe
- Presentación

Mejoras implementadas

Mejoras sobre el esquema

Cambios en los tipos

STATUS

Se cambió de "STATUS" CHAR(100 BYTE), a 10 BYTES para ocupar menos espacio y porque realmente tampoco tenía tanto sentido esto a nivel de lógica de negocios.

```
AlTER TABLE Score
MODIFY Status varchar2(10);
/
```

DATE

Se decidió cambiar el formato TIMESTAMP por DATE, ya que las necesidades del negocio no necesitaban que guardemos segundos en realidad. Esto puede ocasionar una leve mejora en cuanto el espacio de almacenamiento ya que para cada REVIEW hay dos campos de fechas.

Si bien nos parece una modificación pequeña, esta sería la decisión que hubiésemos tomado si hubiéramos diseñado el esquema nosotros.

```
AlTER TABLE EUSER
MODIFY CREATED date;
/
AlTER TABLE EUSER
MODIFY MODIFIED date;
/

AlTER TABLE SCORE
MODIFY CREATED date;
/
AlTER TABLE SCORE
MODIFY MODIFIED date;
```

Cambios en los PCT FREE

Para las tablas EUSER, REVIEW, SALE seteamos los valores en 0 porque solamente hacemos inserciones y no updates, por lo tanto no es necesario reservar espacio para posibles updates en los distintos bloques.

Y por ejemplo si llegamos a hacer una modificación en EUSER, esa modificación no va a cambiar el tamaño del bloque. Los campos de EUSER no tienden a variar.

En el caso de REVIEW, solamente hacemos inserción.
En el caso de SALE, es análogo a REVIEW.
En el caso de SCORE, el caso es distinto porque si va a tener actualizaciones de todas maneras pero no demasiadas, entonces decidimos setearlo en 10.

```
CREATE TABLE "BDII_TEAM3"."SCORE"
( "USER_ID" NUMBER(10,0),
  "LAST_WEEK_POSITIVE" NUMBER(10,0),
  "LAST_WEEK_NEUTRAL" NUMBER(10,0),
  "LAST_WEEK_NEGATIVE" NUMBER(10,0),
  "LAST_MONTH_POSITIVE" NUMBER(10,0),
  "LAST_MONTH_NEUTRAL" NUMBER(10,0),
  "LAST_MONTH_NEGATIVE" NUMBER(10,0),
  "LAST_6MONTH_POSITIVE" NUMBER(10,0),
  "LAST_6MONTH_NEUTRAL" NUMBER(10,0),
  "LAST_6MONTH_NEGATIVE" NUMBER(10,0),
  "SCORE" NUMBER(10,2),
  "STATUS" CHAR(10 BYTE),
  "CREATED" TIMESTAMP (6),
  "MODIFIED" TIMESTAMP (6)
) SEGMENT CREATION IMMEDIATE
PCTFREE 10 PCTUSED 10 INITRANS 1 MAXTRAN
STORAGE(INITIAL 10240 NEXT 10240 MINEXTE
PCTINCREASE 5 FREELISTS 1 FREELIST GROUP
TABLESPACE "BDII_TEAM3_DATA" ;
```

```
ALTER TABLE EUSER PCTFREE 0;
ALTER TABLE REVIEW PCTFREE 0;
ALTER TABLE SALE PCTFREE 0;
```

Primary keys y foreign keys

No se vio la necesidad de modificar ninguna primera key. Paralelamente, si se vio la necesidad de agregar una foreign key.

Primero se creó una foreign key sobre la tabla SCORE y sobre el atributo para que cada score pueda relacionarse con un user. Un foreign key no va a cambiar el performance pero

de esta manera, se relaciona un score con un user. De esta manera, se fuerza que exista un usuario referenciado con cada score.

```
ALTER TABLE SCORE ADD CONSTRAINT "SCORE_EUSER_FK" FOREIGN KEY ("USER_ID")
    REFERENCES "BDII_TEAM2"."EUSER" ("ID") ENABLE;
/
```

Cambios en los índices

Principalmente se crearon los índices para facilitar la búsqueda sobre los datos.

```
CREATE INDEX REVIEW_USER_MODIFIED_IDX
 ON REVIEW ("USER_ID","ROLE", "STATUS", "MODIFIED")
 tablespace team2_indexes;
 /

 CREATE INDEX REVIEW_USER_CREATED_IDX
 ON REVIEW ("USER_ID","CREATED")
 tablespace team2_indexes;
 /
```

Además, se eliminó el índice sobre el atributo CREATED de la tabla REVIEW pues se identificó que no se usaba.

```
drop index REVIEW_STATUS_IDX;
/
```

Index organized table

El cambio se realizó en tabla SCORE. Originalmente se usaba solamente el USER_ID para que cada vez que se insertaba data en la tabla los datos eran ingresados según este valor, pero dado que USER_ID es simplemente un identificador y que no tenía sentido este procesamiento en memoria al momento de ingresar los datos. Se supone que la idea original del desarrollador que implementó esto es para llevar menos tuplas en memoria y que los datos viajen juntos, pero realmente no es necesario hacer esto. Al sacar este “procesamiento” adicional en memoria, es un factor menos que frene el performance.

Mejoras sobre los procedures

Variable vRATIO

Se define vRATIO NUMBER(10,10):=0; y consideramos que esto es un error grave. La razón detrás de esto es que así definido no se pueden representar ni el -1 ni el 1,

solamente los valores que están entre ellos. Como la consulta devuelve valores que pueden llegar a ser -1 y 1, necesitamos poder representarlos.

Antes de la modificación:

```
-- Procedimiento que actualiza la reputacion de un vendedor
PROCEDURE update_seller_scoring( pUserId NUMBER,
    p7Positive NUMBER, p7Negative NUMBER, p7Neutral NUMBER,
    p30Positive NUMBER, p30Negative NUMBER, p30Neutral NUMBER,
    p180Positive NUMBER, p180Negative NUMBER, p180Neutral NUMBER,
    pScoreTotal NUMBER, pCountTotal NUMBER
    ) AS
    vRATIO NUMBER(10,10):=0;
BEGIN
```

Luego de la modificación:

```
-- Procedimiento que actualiza la reputacion de un vendedor
PROCEDURE update_seller_scoring( pUserId NUMBER,
    p7Positive NUMBER, p7Negative NUMBER, p7Neutral NUMBER,
    p30Positive NUMBER, p30Negative NUMBER, p30Neutral NUMBER,
    p180Positive NUMBER, p180Negative NUMBER, p180Neutral NUMBER,
    pScoreTotal NUMBER, pCountTotal NUMBER
    ) AS
    vRATIO NUMBER(3,2):=0;
BEGIN
```

Cursores

Si bien la implementación que nos fue asignada contaba con un cursor de USER_IDs decidimos hacerle una leve modificación que se puede ver a continuación, con el objetivo de verificar esta condición de PUBLISHED antes de hacer los FOR internos que tenía el script. Al mismo tiempo, se sacó esa condición dentro de los FOR internos.

```
-- Cursor de vendedores cuyas calificaciones sufrieron modificaciones
-- o cuya reputacion hay que actualizar porque es "vieja" (de mas de 1 semana)
CURSOR SELLER_CUR( pInterval NUMBER ) IS
SELECT DISTINCT USER_ID
FROM REVIEW
WHERE ROLE = 'SELLER'
AND STATUS = 'PUBLISHED'
AND MODIFIED >= ( SYSDATE - pInterval )
UNION
SELECT USER_ID
FROM SCORE
WHERE MODIFIED >= ( SYSDATE - 7 )
;
```

Además, y consideramos esta la mejora más sustancial en cuanto performance de toda la mejora del script, se agregó un nuevo cursor que trabaja mejor en conjunto con el otro cursor y además elimina la repetición de código. Este cambio si bien fue fácil de identificar porque había mucha lógica repetida. Una parte de esta lógica repetida se puede ver en los siguientes snippets:

```
-- Por cada seller,
FOR SELLER_REC IN SELLER_CUR(ndays) LOOP
  -- Buscar el puntaje recibido en las ventas de la ultima semana
  SELECT COUNT(1)
  INTO v7Positive
  FROM REVIEW
  WHERE USER_ID = SELLER_REC.USER_ID
  AND ROLE = 'SELLER'
  AND CREATED > SYSDATE - 7
  AND STATUS = 'PUBLISHED'
  AND SCORE = 'POSITIVE'
  ;
  SELECT COUNT(1)
  INTO v7Negative
  FROM REVIEW
  WHERE USER_ID = SELLER_REC.USER_ID
  AND ROLE = 'SELLER'
  AND CREATED > SYSDATE - 7
  AND STATUS = 'PUBLISHED'
  AND SCORE = 'NEGATIVE'
  ;
  SELECT COUNT(1)
  INTO v7Neutral
  FROM REVIEW
  WHERE USER_ID = SELLER_REC.USER_ID
  AND ROLE = 'SELLER'
  AND CREATED > SYSDATE - 7
  AND STATUS = 'PUBLISHED'
  AND SCORE = 'NEUTRAL'
  ;
```

```
-- Buscar el puntaje recibido en las ventas del ultimo mes
SELECT COUNT(1)
INTO v30Positive
FROM REVIEW
WHERE USER_ID = SELLER_REC.USER_ID
AND ROLE = 'SELLER'
AND CREATED > SYSDATE - 30
AND STATUS = 'PUBLISHED'
AND SCORE = 'POSITIVE'
;
SELECT COUNT(1)
INTO v30Negative
FROM REVIEW
WHERE USER_ID = SELLER_REC.USER_ID
AND CREATED > SYSDATE - 30
AND ROLE = 'SELLER'
AND STATUS = 'PUBLISHED'
AND SCORE = 'NEGATIVE'
;
SELECT COUNT(1)
INTO v30Neutral
FROM REVIEW
WHERE USER_ID = SELLER_REC.USER_ID
AND CREATED > SYSDATE - 30
AND ROLE = 'SELLER'
AND STATUS = 'PUBLISHED'
AND SCORE = 'NEUTRAL'
;
```

```
-- Buscar el puntaje recibido en las ventas de los ultimos seis meses
SELECT COUNT(1)
INTO v180Positive
FROM REVIEW
WHERE USER_ID = SELLER_REC.USER_ID
AND CREATED > SYSDATE - 180
AND ROLE = 'SELLER'
AND STATUS = 'PUBLISHED'
AND SCORE = 'POSITIVE'
;
SELECT COUNT(1)
INTO v180Negative
FROM REVIEW
WHERE USER_ID = SELLER_REC.USER_ID
AND CREATED > SYSDATE - 180
AND ROLE = 'SELLER'
AND STATUS = 'PUBLISHED'
AND SCORE = 'NEGATIVE'
;
SELECT COUNT(1)
INTO v180Neutral
FROM REVIEW
WHERE USER_ID = SELLER_REC.USER_ID
AND CREATED > SYSDATE - 180
AND ROLE = 'SELLER'
AND STATUS = 'PUBLISHED'
AND SCORE = 'NEUTRAL'
;
```

Y esa lógica terminó siendo usada de la siguiente manera:

```
LOOP
 -- Buscar el puntaje recibido en las ventas de la ultima semana
    FOR REVIEW_REC IN LAST_MONTHS_REVIEWS(SELLER_REC.USER_ID) LOOP
  IF REVIEW_REC.SCORE = 'POSITIVE'  THEN
        v180Positive := v180Positive + 1;
        IF REVIEW_REC.CREATED >  SYSDATE - 30  THEN
            v30Positive := v30Positive + 1;
            IF REVIEW_REC.CREATED >  SYSDATE - 7  THEN
                v7Positive := v7Positive + 1;
            END IF;
        END IF;

    ELSIF REVIEW_REC.SCORE = 'NEUTRAL'  THEN
        v180Neutral := v180Neutral + 1;
        IF REVIEW_REC.CREATED >  SYSDATE - 30  THEN
            v30Neutral := v30Neutral + 1;
            IF REVIEW_REC.CREATED >  SYSDATE - 7  THEN
                v7Neutral := v7Neutral + 1;
            END IF;
        END IF;

    ELSIF REVIEW_REC.SCORE = 'NEGATIVE'  THEN
        v180Negative := v180Negative + 1;
        IF REVIEW_REC.CREATED >  SYSDATE - 30  THEN
            v30Negative := v30Negative + 1;
            IF REVIEW_REC.CREATED >  SYSDATE - 7  THEN
                v7Negative := v7Negative + 1;
            END IF;
        END IF;
    END IF;
END LOOP;
```

Por lo tanto, el nuevo cursor agregado es de REVIEWs:

```
CURSOR LAST_MONTHS_REVIEWS(pUserId NUMBER ) IS
            SELECT CREATED, SCORE
            FROM REVIEW
            WHERE USER_ID = pUserId
            AND CREATED > SYSDATE - 180;
            FOR SELLER_REC IN SELLER_CUR(ndays)
```

En la sección de resultados se puede ver las mejores de performance que logramos introducir gracias a la existencia de este cursor.

Resultados

En cuanto a la naturaleza de los datos proveídos

Los resultados obtenidos en cuanto a performance podrían ser mejores si los datos de la tabla REVIEW en el campo MODIFIED no tuvieran todos el mismo valor. De esta manera los índices propuestos serían más eficientes (para eso se crean los índices) y las búsquedas no darían todas en el mismo lugar.

Se tomó la decisión de no tocar los datos que la empresa provee aunque semánticamente no tengan sentido y muy probablemente hayan sido modificados en algún bulk edit de los registros de la tabla. Si tuvieran una distribución normal por ejemplo, el performance se vería mucho más mejorado. Justamente, cuando hay una mala función hash o una mala distribución de valores a lo largo de una estructura, esto causa problemas de performance.

Resultados experimentales

Todos los resultados experimentales son disparados a partir del siguiente script, el cual, para cada valor de la tabla, se corrió 10 veces y se sacó el valor promedio.

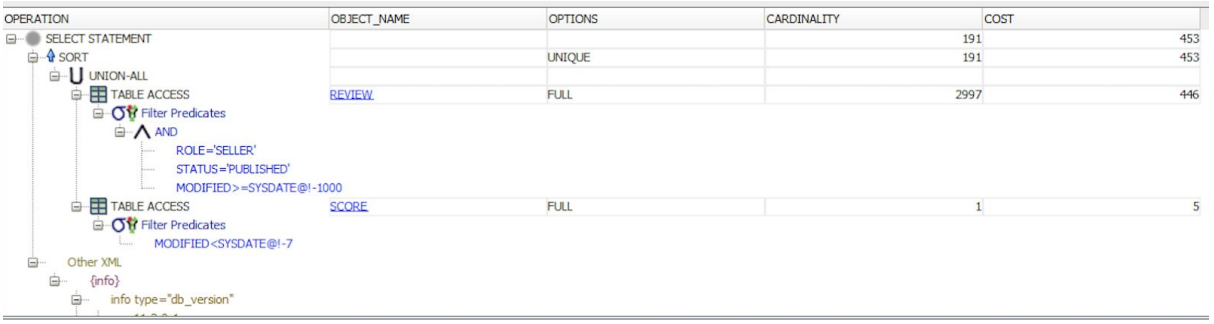
```
DECLARE
  NDAYS NUMBER;
BEGIN
  NDAYS := 1000;

  SCORING_PKG.CALCULATE_SELLER_SCORING(
    NDAYS => NDAYS
  );
--rollback;
END;
/
```

Los valores experimentales obtenidos fueron realizaron sobre las siguientes modificaciones

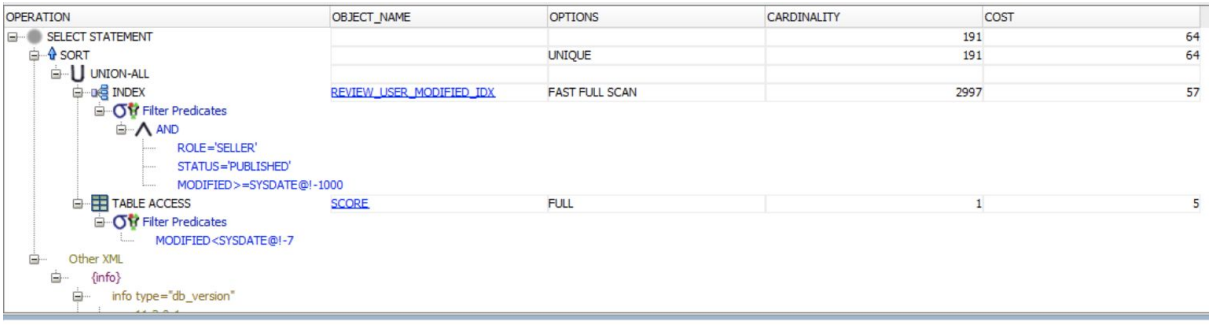
Modificacion	Tiempo (s)
Sin modification	3.918
Nuevo cursor	0.825
Con indices	0.153

Sin uso de índices para el cursor de USER_ID, se puede observar:



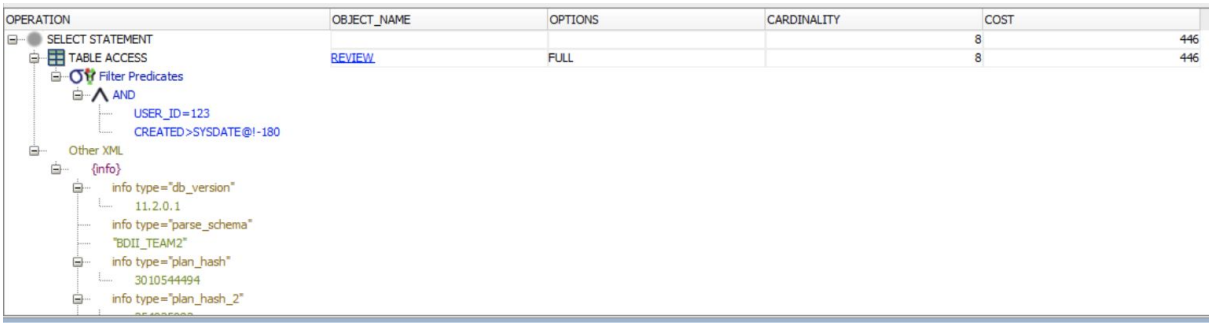
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			191	453
SORT		UNIQUE	191	453
UNION-ALL				
TABLE ACCESS	REVIEW	FULL	2997	446
Filter Predicates				
AND				
ROLE='SELLER'				
STATUS='PUBLISHED'				
MODIFIED>=SYSDATE@!-1000				
TABLE ACCESS	SCORE	FULL	1	5
Filter Predicates				
MODIFIED<SYSDATE@!-7				
Other XML				
{info}				
info type="db_version"				

Con uso de índices para el cursor de USER_ID, se puede observar:



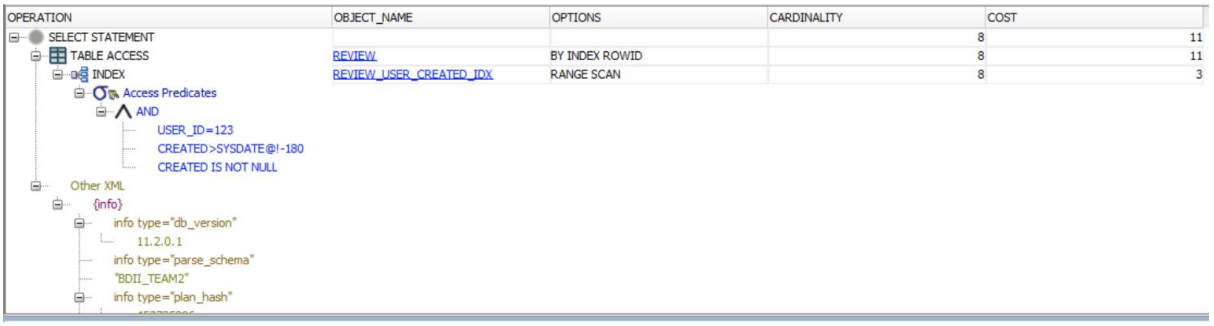
OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			191	64
SORT		UNIQUE	191	64
UNION-ALL				
INDEX	REVIEW_USER_MODIFIED_IDX	FAST FULL SCAN	2997	57
Filter Predicates				
AND				
ROLE='SELLER'				
STATUS='PUBLISHED'				
MODIFIED>=SYSDATE@!-1000				
TABLE ACCESS	SCORE	FULL	1	5
Filter Predicates				
MODIFIED<SYSDATE@!-7				
Other XML				
{info}				
info type="db_version"				

Sin uso de índices para el cursor de REVIEW, se puede observar:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			8	446
TABLE ACCESS	REVIEW	FULL	8	446
Filter Predicates				
AND				
USER_ID=123				
CREATED>SYSDATE@!-180				
Other XML				
{info}				
info type="db_version"				
11.2.0.1				
info type="parse_schema"				
"BDII_TEAM2"				
info type="plan_hash"				
3010544494				
info type="plan_hash_2"				

Con uso de índices para el curso de REVIEW, se puede observar:



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			8	11
TABLE ACCESS	REVIEW	BY INDEX ROWID	8	11
INDEX	REVIEW_USER_CREATED_IDX	RANGE SCAN	8	3
Access Predicates				
AND				
USER_ID=123				
CREATED>SYSDATE@!-180				
CREATED IS NOT NULL				
Other XML				
{info}				
info type="db_version"				
11.2.0.1				
info type="parse_schema"				
"BDII_TEAM2"				
info type="plan_hash"				

Conclusión

Hubo principalmente dos tipos de mejoras en este proyecto.

En primera medida, la mayoría de las mejoras sobre el esquema de la base de datos conciernen el espacio usado, y llevan más sentido en la organización de la base de datos, esto hace que todas las queries y organización de la información tengan más sentido semántico.

Por otra parte, las mejoras sobre los procedures se pueden ver en que la velocidad aumentó como se puede ver en la primer tabla (disminuyeron los tiempos de ejecución) y también se puede ver que hubo mejoras en el performance como se puede ver en estas últimas cuatro imágenes (gracias al mejor uso de índices). Aunque la naturaleza de los datos proveídos no permite comprobar la mejora del performance brindada por el índice sobre el campo *modified* de la tabla review, que se usa para el cursor USER_ID en la procedura, vemos que el *cost* dado por el *query execution plan* disminuyó considerablemente con este nuevo índice, así que con una distribución de datos repartida de manera más uniforme podemos suponer que el tiempo de ejecución se volvería aún mejor.