

Sistemas Informáticos

Tema 3: Bases de datos distribuidas

3.5 Optimización de consultas



Explain plan

- Cada vez que ejecutamos una sentencia SQL en el SGBD éste crea el plan de ejecución (*explain plan*) de la sentencia
- El explain plan define la forma en que el SGBD busca o inserta los datos:
 - Formas de cruzar tablas
 - Uso de índices
 - Orden de ejecución de subconsultas
 - ...
- La información suministrada por el explain plan permite identificar cuellos de botella en la ejecución de una consulta
 - Índices
 - Cruces y estructura de sentencias



Explain plan en MySQL (I)

- Analizamos en detalle el caso de MySQL, pero hay comandos equivalentes en prácticamente todos los SGBD (ver más adelante ejemplos de PostgreSQL)
- EXPLAIN devuelve el plan de acceso de una sentencia SQL en forma de tabla
 - Cada fila contiene información de las tablas (físicas o no) empleadas en la consulta
 - El orden de las tablas indica el orden en el que se procesarían en la consulta (a tener en cuenta de cara a la optimización)
 - SHOW WARNINGS → optimizador



Explain plan en MySQL (II)

```

1 • EXPLAIN EXTENDED SELECT
2   nombre_cliente
3
4   FROM
5   cliente,
6   (SELECT
7     id_cliente
8   FROM
9     prestamo, cliente_prestamo
10    WHERE
11      prestamo.numero_prestamo = cliente_prestamo.numero_prestamo
12      AND nombre_sucursal = 'Perryridge'
13      AND cliente_prestamo.id_cliente NOT IN (SELECT
14        id_cliente
15      FROM
16        cuenta, cliente_cuenta
17      WHERE
18        cuenta.numero_cuenta = cliente_cuenta.numero_cuenta
19        AND nombre_sucursal = 'Perryridge')) AS tabla_aux
20  WHERE
21    id_cliente = id;
  
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	prestamo		ALL	PRIMARY,numero_prestamo				9	11.11	Using where
1	PRIMARY	cliente_prestamo		ref	PRIMARY,numero_prestamo	numero_prestamo	47	banco.prestamo.numero_prestamo	1	100.00	Using where; Using index
1	PRIMARY	cliente		eq_ref	PRIMARY	PRIMARY	47	banco.cliente_prestamo.id_cliente	1	100.00	
3	DEPENDENT SUBQUERY	cliente_cuenta		ref	PRIMARY,numero_cuenta	PRIMARY	47	func	1	100.00	Using index
3	DEPENDENT SUBQUERY	cuenta		eq_ref	PRIMARY	PRIMARY	47	banco.cliente_cuenta.numero_cuenta	1	11.11	Using where

- type → indica el tipo de cruce de las tabla
 - Tratar de evitar ALL
 - const, eq_ref, ref



BD de prueba

- Trabajaremos con la base de datos de películas disponible en Moodle
- Añadiremos además una nueva tabla

```
CREATE TABLE muchasPeliculas(
    ID INTEGER,          -- Identificador único
    TITULO CHAR(128),    -- Título de la película
    AGNO DECIMAL(4),     -- Año de estreno
    PUNTUACION FLOAT,    -- Puntuación media
    VOTOS INTEGER,       -- Número de votos
);
```

- Antes del explain plan hay otras elementos a tener en cuenta...



Rendimiento creación y carga

CREATE TABLE PELICULA(ID INTEGER, -- Identifica...	0 row(s) affected	0,180 sec
CREATE TABLE ACTOR (ID INTEGER, -- Identificador U...	0 row(s) affected	0,340 sec
CREATE TABLE REPARTO(PELICULA_ID INTEGER, -- refer...	0 row(s) affected	0,215 sec
INSERT INTO PELICULA (ID, TITULO, AGNO, PUNTUACION, VOT...	1686 row(s) affected Records: 1686 Duplicates: 0 Warnings: 0	0,364 sec
COMMIT	0 row(s) affected	0,00099 sec
INSERT INTO ACTOR (ID, NOMBRE) VALUES (1,'Ángel Sala...	9119 row(s) affected Records: 9119 Duplicates: 0 Warnings: 0	1,263 sec
COMMIT	0 row(s) affected	0,00099 sec
6926,35), (1166,6926,49), (1353,6926,41), (233,69	30183 row(s) affected Records: 30183 Duplicates: 0 Warnings: 0	2,053 sec



Rendimiento creación y carga

```
INSERT INTO REPARTO VALUES (280,1,10);  
INSERT INTO REPARTO VALUES (503,1,15);  
...  
INSERT INTO REPARTO VALUES (713,9119,19);  
INSERT INTO REPARTO VALUES (1592,9119,22);
```

Rendimiento: 780 s = 13 m
Desactivando primary key y foreing keys: 782 s

```
INSERT INTO REPARTO (PELICULA_ID, ACTOR_ID, ORD) VALUES  
  (280,1,10),  
  (503,1,15),  
...  
  (713,9119,19),  
  (1592,9119,22);
```

Rendimiento: 2 s



Generando volumen de datos

- Copia de la tabla de películas pero con muchas películas

```
CREATE PROCEDURE llenado_películas(IN n INT)  
BEGIN  
  DECLARE maxID INT DEFAULT 1;  
  DECLARE i INT DEFAULT 1;  
  
  SELECT max(ID) INTO maxID FROM PELICULA;  
  WHILE i < n DO  
    INSERT INTO muchasPelículas  
      (ID, TITULO, AGNO, PUNTUACION, VOTOS)  
      (SELECT ((i-1)*maxID) + ID,  
        CONCAT(TITULO, ' v', i),  
        AGNO,  
        ROUND(RAND()*10,2),  
        FLOOR(RAND()*1000)  
      FROM PELICULA);  
    SET i = i + 1;  
  END WHILE;  
END
```



BD ejemplo

1
2 • `SELECT * FROM muchasPelículas;`

Result Grid


#	ID	TITULO	AGNO	PUNTUACION	VOTOS
1	1	Star Wars v1	1977	0.51	906
2	2	Pulp Fiction v1	1994	3.76	160
3	3	Blade Runner v1	1982	6.77	902
4	4	Titanic v1	1997	4.82	701
5	5	Braveheart v1	1995	0.6	195
6	6	Empire Strikes Back, The v1	1980	8	413
7	7	Shawshank Redemption, The v1	1994	6.66	91
8	8	Independence Day v1	1996	4.57	10
9	9	Usual Suspects, The v1	1995	6.84	387
10	10	Raiders of the Lost Ark v1	1981	8.87	272
11	11	2001: A Space Odyssey v1	1968	7	681
12	12	Forrest Gump v1	1994	3.07	493

muchasPelículas 1

Action Output

	Time	Message	Duration / Fetch
1	18:36	>5000000 5056314 row(s) returned	0.0016 sec / 6,426 sec

???



SELECT

`SELECT * FROM muchasPelículas;`

Rendimiento: 0.0016 segundos

`SELECT count(*) FROM muchasPelículas;`

Rendimiento : 1.653 segundos

`SELECT * FROM muchasPelículas where ID = 235719;`

Rendimiento : 2.172 segundos

`SELECT * FROM muchasPelículas where ID = 23571900;`

Rendimiento : 2.169 segundos

Query cost: 1029690.60

query_block #1

1029690.60 4.89M rows

Full Table Scan

muchasPelículas



Creamos un índice implícito

```
ALTER TABLE muchasPelículas ADD PRIMARY KEY(ID)
```

Rendimiento: 105.074 segundos



Comparativa

```
SELECT * FROM muchasPelículas;
```

Rendimiento: 0.0013 segundos

```
SELECT count(*) FROM muchasPelículas;
```

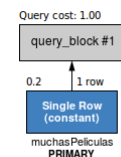
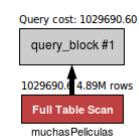
Rendimiento : 1.147 segundos

```
SELECT * FROM muchasPelículas where ID = 235719;
```

Rendimiento : 0.00050 segundos

```
SELECT * FROM muchasPelículas where ID = 23571900;
```

Rendimiento : 0.00042 segundos



Creamos un índice explícito

```
CREATE INDEX idx_id_pelis ON muchasPelículas(ID)
```

Rendimiento: 24.787 segundos



SELECT

```
SELECT * FROM muchasPelículas;
```

Rendimiento: 0.0016 segundos

```
SELECT count(*) FROM muchasPelículas;
```

Rendimiento : 1.250 segundos

```
SELECT * FROM muchasPelículas where ID = 235719;
```

Rendimiento : 0.00043 segundos

```
SELECT * FROM muchasPelículas where ID = 23571900;
```

Rendimiento : 0.00049 segundos

Query cost: 1029690.60
query_block #1
1029690.6 4.89M rows
Full Table Scan
muchasPelículas

Query cost: 1.20
query_block #1
1.2 1 row
Non-Unique Key Lookup
muchasPelículas
idx_id_pelis



SELECT

```
SELECT * FROM muchasPelículas where  
TITULO = 'Indiana Jones and...';
```

Rendimiento: 2.300 segundos

```
SELECT * FROM muchasPelículas where  
TITULO = 'Fallido';
```

Rendimiento : 2.287 segundos

```
SELECT * FROM muchasPelículas where  
TITULO LIKE 'Indiana %';
```

Rendimiento : 0.095 segundos

```
SELECT * FROM muchasPelículas where  
TITULO LIKE '%Jones %';
```

Rendimiento : 0.120 segundos

```
SELECT * FROM muchasPelículas where  
TITULO LIKE 'Fallido %';
```

Query cost: 1029690.60

query_block #1

1029690.60 4.89M rows

Full Table Scan

muchasPelículas



Creamos un índice explícito


```
CREATE INDEX idx_name_pelis ON muchasPelículas(TITULO)
```

Rendimiento: 114.697 segundos



SELECT

<pre>SELECT * FROM muchasPeliculas where TITULO = 'Indiana Jones and...';</pre> <p style="color: green;">Rendimiento: 0.00073 segundos</p>	<p>Query cost: 1.20</p> <p>query_block #1</p> <p>1.2 1 row</p> <p>Non-Unique Key Lookup</p> <p>muchasPeliculas idx_name_pells</p>
<pre>SELECT * FROM muchasPeliculas where TITULO = 'Fallido';</pre> <p style="color: green;">Rendimiento : 0.00055 segundos</p>	
<pre>SELECT * FROM muchasPeliculas where TITULO LIKE 'Indiana %';</pre> <p style="color: green;">Rendimiento : 0.0036 segundos</p>	<p>Query cost: 16663.81</p> <p>query_block #1</p> <p>16663.81 11.90K rows</p> <p>Index Range Scan</p> <p>muchasPeliculas idx_name_pells</p>
<pre>SELECT * FROM muchasPeliculas where TITULO LIKE 'Fallido %';</pre> <p style="color: green;">Rendimiento : 0.0037 segundos</p>	
<pre>SELECT * FROM muchasPeliculas where TITULO LIKE '%Jones %';</pre> <p style="color: green;">Rendimiento : 0.121 segundos</p>	<p>Query cost: 1029690.60</p> <p>query_block #1</p> <p>1029690.60 4.89M rows</p> <p>Full Table Scan</p> <p>muchasPeliculas</p>

 EPS

Recálculo de estadísticas

- Después de cada actualización grande (sobre todo, muchas inserciones / borrados) se debe actualizar la información que el motor tiene sobre el contenido de las tablas.
- Por ejemplo:
 - En MySQL:
 - ANALYZE TABLE
 - En PostgreSQL
 - ANALYZE

EXPLAIN PLAN en PostgreSQL (I)

- EXPLAIN nos permite "ver" como planea postgresql ejecutar una consulta

```
explain SELECT * FROM muchasPelículas where title = 'REQUIEM TYCOONv236';
```

QUERY PLAN

```
-----
"Seq Scan on muchaspelículas (cost=0.00..706901.80 rows=4 width=150)"
"  Filter: (title = 'REQUIEM TYCOONv236'::text)"
```

Lee toda la
tabla

```
explain SELECT * FROM muchasPelículas where id_film = 235719;
```

QUERY PLAN

```
-----
"Index Scan using idx_id_pelis on muchaspelículas (cost=0.00..17.66 rows=4 width=150)"
"  Index Cond: (id_film = 235719)"
```

Usa el
Índice



EXPLAIN PLAN en PostgreSQL (II)

```
Seq Scan on muchaspelículas (cost=0.00..706901.80 rows=1 width=20)
Filter: (numero = 65985828)
```

relación

condición, para que hace el Seq. Scan

- El primer numero es una estimación del tiempo necesario antes de que la primera tupla pueda ser obtenida.
- El segundo número es una estimación del tiempo TOTAL.
- Estimación del número de filas devueltas
- Estimación del numero de bytes por fila

tipo búsqueda



EXPLAIN PLAN en PostgreSQL (III)

```
EXPLAIN SELECT id, COUNT(*) FROM test GROUP BY id;
QUERY PLAN
```

```
Aggregate (cost=0.00..122799.00 rows=500000 width=4)
-> Group (cost=0.00..110299.00 rows=5000000 width=4)
    -> Index Scan using idx_id_testdb on test
        (cost=0.00..97799.00 rows=5000000 width=4)
```

(3 rows)

La base tiene que realizar tres operaciones "agregación", "agrupamiento" y lectura del índice (versus lectura de la tabla)



EXPLAIN PLAN en PostgreSQL (IV)

```
EXPLAIN SELECT numero, COUNT(*) FROM test GROUP BY numero;
QUERY PLAN
```

```
Aggregate (cost=888535.42..926035.42 rows=500000 width=4)
-> Group (cost=888535.42..913535.42 rows=5000000 width=4)
    -> Sort (cost=888535.42..901035.42 rows=5000000 width=4)
        Sort Key: numero
    -> Seq Scan on test (cost=0.00..81848.00 rows=5000000 width=4)
        (5 rows)
```

4) Con índices no hace falta

1) Hay que leer TODA la tabla

2) El índice no nos aporta ganancia en este paso

3) Pero a la hora de ordenar



EXPLAIN PLAN en PostgreSQL (V)

```
EXPLAIN SELECT * FROM muchasPeliculas;
```

```
"Seq Scan on muchaspeliculas  
(cost=0.00..164233.72 rows=4999872 width=150)"
```

~16 segs VS 15 segs



EXPLAIN PLAN en PostgreSQL (VI)

```
EXPLAIN SELECT * FROM muchasPeliculas where id_film =  
235719;
```

```
"Index Scan using idx_id_pelis on muchaspeliculas  
(cost=0.00..8.82 rows=1 width=150)" "  
Index Cond: (id_film = 235719)"
```

~1580 msecs VS 18 msecs



EXPLAIN PLAN en PostgreSQL (VII)

```
SELECT * FROM muchasPeliculas where title = 'REQUIEM TYCOONv236'
```

```
"Index Scan using idx_name_pelis on muchaspeliculas  
(cost=0.00..9.25 rows=1 width=150)" "  
Index Cond: (title = 'REQUIEM TYCOONv236'::text)"
```

~1700 msecs VS 18 msecs



EXPLAIN PLAN en PostgreSQL (VII)

```
SELECT * FROM muchasPeliculas where title LIKE 'REQUIEM %'
```

```
"Seq Scan on muchaspeliculas  
(cost=0.00..176733.40 rows=500 width=150)" "  
Filter: (title ~~ 'REQUIEM %'::text)"
```

~1800 msecs VS 1800 msecs



EXPLAIN PLAN en PostgreSQL (IX)

```
SELECT * FROM muchasPeliculas where year = 1995
```

```
Bitmap Heap Scan on muchasPeliculas
(cost=3036.27..125182.24 rows=161996 width=150)
  Recheck Cond: (year = 1995)
    -> Bitmap Index Scan on idx_year_pelis
        (cost=0.00..2995.77 rows=161996 width=0)
          Index Cond: (year = 1995)"
```

~2100 msecs VS 1900 msecs



EXPLAIN PLAN en PostgreSQL (X)

```
SELECT * FROM muchasPeliculas where year = 1995
```

~2100 msecs VS 1900 msecs

