Veritabanı Yönetim Sistemleri

(Başarım Eniyileme - Performance Tuning)

Konular

- ✓ SQL Başarım Eniyileme (SQL Performance Tuning)

EXPLAIN ANALYSE

✓ **EXPLAIN ANALYSE** ifadesi ile SQL sorgularının başarımına ilişkin detaylı bilgi edinebiliriz.

```
EXPLAIN ANALYSE
SELECT * FROM "orders"
WHERE "ShipCity" = 'Bern';
```

	QUERY PLAN
1	Seq Scan on orders (cost=0.00..24.38 rows=8 width=90) (actual time=0.012..0.234 rows=8 loops=1)
2	Filter: (("ShipCity")::text = 'Bern'::text)
3	Rows Removed by Filter: 822
4	Planning time: 0.067 ms
5	Execution time: 0.261 ms

PROJEKSİYON

- ✓ **SELECT** ifadesinde bütün alanlara projeksiyon yapmak (* kullanımı) yerine yalnızca gerekli olan alanlara projeksiyon yapmalıyız.

- ✓ Yani yalnızca gerekli alanların bilgilerini göstermeliyiz.

```
EXPLAIN ANALYSE
SELECT *
FROM "customer"
INNER JOIN "store" ON "customer"."store_id" = "store"."store_id"
INNER JOIN "rental" ON "rental"."customer_id" = "customer"."customer_id"
INNER JOIN "inventory" ON "inventory"."store_id" = "store"."store_id"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_id";

20:52:27 Query time: 12.442 second(s), Number of cursor's records: 23


EXPLAIN ANALYSE
SELECT "customer"."first_name", "customer"."last_name",
    "film"."film_id", "film"."title"
FROM "customer"
INNER JOIN "store" ON "customer"."store_id" = "store"."store_id"
INNER JOIN "rental" ON "rental"."customer_id" = "customer"."customer_id"
INNER JOIN "inventory" ON "inventory"."store_id" = "store"."store_id"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_id";

20:52:40 Query time: 7.177 second(s), Number of cursor's records: 24
```

LIMIT ve OFFSET

- ✓ **LIMIT** ve **OFFSET** kullanımı, sorgularımızı hızlandırır.

```
EXPLAIN ANALYSE
SELECT "store"."store_id", "film"."title"
FROM "inventory"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_id"
INNER JOIN "store" ON "inventory"."store_id" = "store"."store_id";

21:04:05 Query time: 5 millisecond(s), Number of cursor's records: 13
```

```
EXPLAIN ANALYSE
SELECT "store"."store_id", "film"."title"
FROM "inventory"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_id"
INNER JOIN "store" ON "inventory"."store_id" = "store"."store_id"
LIMIT 20  OFFSET 39;

21:04:16 Query time: 2 millisecond(s), Number of cursor's records: 12
```

SIRALAMA

✓ Gereksiz sıralama başarımı düşürür.

```
EXPLAIN ANALYSE
SELECT "store"."store_id", "film"."title"
FROM "inventory"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_id"
INNER JOIN "store" ON "inventory"."store_id" = "store"."store_id";

21:12:37 Query time: 5 millisecond(s), Number of cursor's records: 13
```

```
EXPLAIN ANALYSE
SELECT "store"."store_id", "film"."title"
FROM "inventory"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_id"
INNER JOIN "store" ON "inventory"."store_id" = "store"."store_id"
ORDER BY "film"."title";

21:12:41 Query time: 7 millisecond(s), Number of cursor's records: 16
```

INDEX

- ✓ Index olarak belirlenmiş alanlar üzerinde arama işlemi daha hızlı gerçekleştirilir.
- ✓ Aşağıdaki sorgularda “customer” tablosunun “last_name” alanı index olarak belirlenmiştir.

```
EXPLAIN ANALYSE
SELECT * FROM "customer"
WHERE "first_name" = 'Jeniffer';

Execution time: 0.132 ms
```

```
EXPLAIN ANALYSE
SELECT * FROM "customer"
WHERE "last_name" = 'Davis';

Execution time: 0.036 ms
```

EXISTS ve IN

- ✓ **IN** ifadesinin başarımı genellikle düşüktür.
- ✓ Filtreleme kriterlerinin çoğu ***altsorguda*** ise **IN** kullanımı etkilidir.
- ✓ Filtreleme kriterlerinin çoğu ***anasorguda*** ise **EXISTS** kullanımı etkilidir.

EXISTS ve IN

```
EXPLAIN ANALYSE
SELECT DISTINCT "customer"."first_name", "customer"."last_name"
FROM "customer"
WHERE "customer_id" IN (SELECT "customer_id" FROM "payment");

21:34:45 Query time: 3 millisecond(s), Number of cursor's records: 9


EXPLAIN ANALYSE
SELECT "customer"."first_name", "customer"."last_name"
FROM "customer"
WHERE "customer_id" IN (SELECT DISTINCT "customer_id" FROM "payment");

21:34:49 Query time: 6 millisecond(s), Number of cursor's records: 10


EXPLAIN ANALYSE
SELECT "customer"."first_name", "customer"."last_name"
FROM "customer"
WHERE EXISTS
   (SELECT "customer_id" FROM "payment"
    WHERE "customer"."customer_id" = "payment"."customer_id");

21:34:53 Query time: 4 millisecond(s), Number of cursor's records: 7
```

BİRLEŞİM

✓ Birleşim kullanımı bire-çok ilişkiye sahip tabloları kapsıyorsa **DISTINCT** yerine **EXISTS** kullanınız.

```
EXPLAIN ANALYSE
SELECT DISTINCT "customer"."first_name", "customer"."last_name"
FROM "customer”
INNER JOIN "payment"ON "payment"."customer_id" = "customer"."customer_id";

21:47:16 Query time: 14 millisecond(s), Number of cursor's records: 10


EXPLAIN ANALYSE
SELECT "customer"."first_name", "customer"."last_name"
FROM "customer"WHERE EXISTS
   (SELECT * FROM "payment”
    WHERE "payment"."customer_id" = "customer"."customer_id");

21:47:19 Query time: 4 millisecond(s), Number of cursor's records: 7
```

HAVING

- ✓ **HAVING** ifadesi seçim işlemi yapılıp gruplandırma işlemi tamamlandıktan sonra filtreleme yapmak için kullanmalıyız.

```
EXPLAIN ANALYSE
SELECT "category"."name", COUNT("film"."film_id")
FROM "film"
LEFT OUTER JOIN "film_category" ON "film"."film_id" = "film_category"."film_id"
LEFT OUTER JOIN "category" ON "film_category"."category_id" =
    "category"."category_id"
GROUP BY "category"."name"
HAVING "category"."name" = 'Horror' OR "category"."name" = 'Comedy';

22:04:45 Query time: 3 millisecond(s), Number of cursor's records: 16
```

```
EXPLAIN ANALYSE
SELECT "category"."name", COUNT("film"."film_id")
FROM "film"
LEFT OUTER JOIN "film_category" ON "film"."film_id" = "film_category"."film_id"
LEFT OUTER JOIN "category" ON "film_category"."category_id" =
    "category"."category_id"
WHERE "category"."name" = 'Horror' OR "category"."name" = 'Comedy'
GROUP BY "category"."name";

22:05:02 Query time: 2 millisecond(s), Number of cursor's records: 16
```

Alt Sorgu Sayısı

- ✓ Bazen ana sorguda birden fazla alt sorgu bulunabilir.
- ✓ Bu durumda alt sorgu bloklarının sayısını azaltmaya çalışmalıyız.

```
EXPLAIN ANALYSE
SELECT * FROM "products"
WHERE "UnitPrice" < (SELECT AVG("UnitPrice") FROM "products")
AND "UnitsInStock" < (SELECT AVG("UnitsInStock") FROM "products");

22:12:27 Query time: 2 millisecond(s), Number of cursor's records: 11
```

```
EXPLAIN ANALYSE
SELECT * FROM "products"
WHERE ("UnitPrice", "UnitsInStock") <
    (SELECT AVG("UnitPrice"), AVG("UnitsInStock") FROM "products");

22:12:32 Query time: 1 millisecond(s), Number of cursor's records: 8
```

UNION ve UNINON ALL

- ✓ **UNION** yerine **UNION ALL** komutunu kullanmaya çalışmalıyız.
- ✓ **UNION** komutu icra edilirken **DISTINCT** işlemi de gerçekleştirildiği için daha yavaştır

```
EXPLAIN ANALYSE
SELECT "rental_id" FROM "rental"
UNION
SELECT "rental_id" FROM "payment";

22:23:50 Query time: 21 millisecond(s), Number of cursor's records: 7
```

```
EXPLAIN ANALYSE
SELECT "rental_id" FROM "rental"
UNION ALL
SELECT "rental_id" FROM "payment";

22:23:53 Query time: 11 millisecond(s), Number of cursor's records: 5
```

WHERE

✓ **WHERE** koşul ifadeleri yazarken dikkat etmemiz gereken hususlar.

```
EXPLAIN ANALYSE
SELECT * FROM "payment" WHERE "amount" != 11.99;

22:38:13 Query time: 6 millisecond(s), Number of cursor's records: 5


EXPLAIN ANALYSE
SELECT * FROM "payment" WHERE "amount" < 11.99;

22:38:13 Query time: 6 millisecond(s), Number of cursor's records: 5
```

WHERE

✓ **WHERE** koşul ifadeleri yazarken dikkat etmemiz gereken hususlar.

```
EXPLAIN ANALYSE
SELECT * FROM "film" WHERE SUBSTR("title", 2, 2) = 'la';

22:44:30 Query time: 2 millisecond(s), Number of affected records: 15


EXPLAIN ANALYSE
SELECT * FROM "film" WHERE "title" LIKE '_la%';

22:44:25 Query time: 1 millisecond(s), Number of affected records: 15
```

WHERE

✓ **WHERE** koşul ifadeleri yazarken dikkat etmemiz gereken hususlar.

```
EXPLAIN ANALYSE
SELECT * FROM "customer"
WHERE "first_name" || "last_name" = 'LisaAnderson';

22:48:11 Query time: 2 millisecond(s), Number of affected records: 1


EXPLAIN ANALYSE
SELECT * FROM "customer"
WHERE "first_name" = 'Lisa' AND "last_name" = 'Anderson';

22:48:15 Query time: 1 millisecond(s), Number of affected records: 1
```

WHERE

✓ **WHERE** koşul ifadeleri yazarken dikkat etmemiz gereken hususlar.

```
EXPLAIN ANALYSE
SELECT * FROM "payment"
WHERE "amount" - 1 = '1.99';

22:50:54 Query time: 6 millisecond(s), Number of cursor's records: 5


EXPLAIN ANALYSE
SELECT * FROM "payment"
WHERE "amount" = '2.99';

22:50:56 Query time: 5 millisecond(s), Number of cursor's records: 5
```

Genel Kurallar

- ✓ Büyük ikili nesneleri depolamak için ilk önce onları dosyalama sistemine yerleştiriniz ve veritabanına dosyanın konumunu ekleyiniz.
- ✓ Etkin performans sağlayan SQL sorguları yazmak için genel **SQL** standart kurallarını takip ediniz.

VACUUM

- ✓ PostgreSQL'de bir kayıt silindiği zaman aslında gerçekten silinmez.
- ✓ Yalnızca silindiğine ilişkin bir işaret olur.
- ✓ Dolayısıyla belli bir süre sonra depolama alanı problemi oluşabilir.
- ✓ Silinen kayıtların gerçekten tablodan silinmesini gerçekleştirmek için VACUUM komutu kullanılır.
- ✓ Bu yapıldığında depolama alanımızda yer açılacaktır.

```
VACUUM; -- Seçili veri tabanındaki tüm tabloları vakumla.
VACUUM FULL; -- Daha fazla yer aç. Daha uzun sürer.
VACUUM customer; -- customer tablosunu vakumla.

-- Threshold değerini %20 aştıktan sonra otomatik vakum işlemi yap.
-- Varsayılan 0.2
ALTER TABLE table_name
SET (autovacuum_vacuum_scale_factor = 0.3);


-- Threshold değeri 5000 kayıt olsun.
-- Varsayılan 50 kayıt.
ALTER TABLE table_name
SET (autovacuum_vacuum_threshold = 5000);
```

VACUUM

```
SELECT "relname", "last_vacuum", "last_autovacuum", "last_analyze",
"last_autoanalyze”
FROM "pg_stat_all_tables”
WHERE "schemaname" = 'public';
```

	relname	last_vacuum	last_autovacuum	last_analyze	last_autoanalyze
1	payment	<NULL>	<NULL>	<NULL>	2016-11-27 18:08:49.244...
2	customer	<NULL>	<NULL>	<NULL>	2016-11-27 18:08:49.34...
3	film_category	<NULL>	<NULL>	<NULL>	2016-11-27 18:08:49.055...
4	film_actor	<NULL>	<NULL>	<NULL>	2016-11-27 18:08:49.44...
5	address	<NULL>	<NULL>	<NULL>	2016-11-27 18:08:49.399...
6	category	<NULL>	<NULL>	<NULL>	<NULL>

Kaynaklar

- ✓ http://www.postgresql.org/docs/9.5/static/sql-vacuum.html
- ✓ http://www.postgresql.org/docs/current/static/using-explain.html