YILDIZ TEKNİK ÜNİVERSİTESİ
ELEKTRİK ELEKTRONİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
VERİ TABANI SİSTEMLERİNDE GÜNCEL KONULAR

İKİNCİ ÖDEVİ

Ahmet Akib Gültekin – 20011068

Muhammet Kayra Bulut – 20011901

Senaryo 1

İlk olarak ;

```
CREATE TABLE accounts (
    id SERIAL PRIMARY KEY,
    client TEXT,
    amount NUMERIC
);

INSERT INTO accounts (client, amount) VALUES ('alice', 1000.00), ('bob', 100.00), ('bob', 900.00);
```

Komutlarını çalıştırarak ilgili tabloyu oluşturup, istenen verileri yükledik. Sonrasında;

a1-)

```
BEGIN;
UPDATE accounts SET amount = amount - 200 WHERE client = 'alice';
SELECT amount FROM accounts WHERE client = 'alice';
```

<H1,H2> işlemlerini sırayla çalıştırdık ve aşağıdaki çıktıyı aldık.

	amount numeric
1	800

<H1, H2> çalışması sonucunda T tablosunun son durumu:

<1, 'alice', 800.00>: H1 hareketi hesaptan 200 lira çektiği için bakiye 800.00 olarak güncellendi.

H2 hareketi, H1 hareketini tamamladıktan sonra hesap bakiyesini 800.00 olarak görüntüledi.

a2-)

Sonrasında, tablodaki verileri eski haline döndürdük ve ardından;

```
BEGIN;
SELECT amount FROM accounts WHERE client = 'alice';
UPDATE accounts SET amount = amount - 200 WHERE client = 'alice';
```

<H2,H1> işlemlerini sırayla çalıştırdık ve aşağıdaki çıktıyı aldık.

	amount numeric
1	1000

<H2, H1> çalışması sonucunda T tablosunun son durumu:

<1, 'alice', 1000.00>: İlk olarak H2 hareketi çalıştığı için hesap bakiyesi 1000.00 olarak görüntülenir.

H1 hareketi, H2 hareketinden sonra hesaptan 200 lira çeker ve bakiyeyi 800.00 olarak günceller.

b1-)

H1 işlemini commit etmeden çalıştırmak için;

```
-- H1 hareketi: 'alice' hesabından 200 lira çekme
BEGIN;
UPDATE accounts SET amount = amount - 200 WHERE client = 'alice';
```

Komutunu kullandık.

Sonrasında H2 işlemini commit etmeden çalıştırmak için;

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SELECT amount FROM accounts WHERE client = 'alice';
```

Komutunu kullandık ve;

	amount numeric
1	1000

Çıktısını aldık.

b2-)

Sonrasında H1 işlemini commit edip tekrardan H2 işlemini çalıştırdığımızda;

	amount numeric
1	800

Çıktısını aldık.

Bu çıktıları almamızın nedeni **READ COMMITTED** yalıtım seviyesinde, bir işlem (transaction) içinde yapılan değişiklikler diğer işlemler tarafından görülebilir, ancak henüz commit edilmemiş değişiklikler geçerli değildir. İlk olarak H1 hareketi gerçekleştirildiğinde, H2 hareketi başlangıç bakiyesini gösterir. H1 hareketi commit edildikten sonra, H2 hareketi güncel bakiyeyi gösterir. **READ COMMITTED** yalıtım seviyesinde, işlemler arasındaki değişiklikler anlık olarak görülmez, sadece commit edilen değişiklikler görünür hale gelir.

c1-)

H1 işlemini commit etmeden çalıştırmak için;

```
-- H1 hareketi: 'alice' hesabından 200 lira çekme
BEGIN;
UPDATE accounts SET amount = amount - 200 WHERE client = 'alice';
```

Komutunu kullandık.

Sonrasında H2 işlemini commit etmeden **REPEATABLE READ** modda çalıştırmak için;

```
-- H2 hareketi: 'alice' hesabını sorgulama
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT amount FROM accounts WHERE client = 'alice';
```

Komutunu kullandık ve;

	amount numeric
1	1000

Çıktısını aldık.

Sonrasında H1 işlemini commit edip tekrardan H2 işlemini çalıştırdığımızda;

	amount numeric
1	1000

Çıktısını aldık.

İlk çıktıyı almamızın nedeni H2 için yazdığınız sorgu, H1 hareketi henüz commit edilmediği için başlangıç bakiyesini (1000.00) gösterecektir. **REPEATABLE READ** yalıtım seviyesinde, bir işlem (transaction) içinde yapılan değişiklikler diğer işlemler tarafından görülemez. H1 hareketi içindeki değişiklikler, H2 hareketi içinde bile görülemez. Bu nedenle, H2 hareketi başlangıç bakiyesini gösterecektir. Sonraki çıktı içinse, H1 hareketi commit edildiğinde, değişiklikler kalıcı hale gelir. Ancak, **REPEATABLE READ** yalıtım seviyesinde işlem içinde okunan verilerin başka bir işlem tarafından değiştirilmesi engellenir. Bu nedenle, H2 hareketini tekrar çalıştırdığınızda başlangıç bakiyesi (1000.00) görüntülenecektir.

Senaryo 2

İlk olarak;

```
CREATE TABLE T (
    id SERIAL PRIMARY KEY,
    name TEXT
);

INSERT INTO T (id, name)
SELECT generate_series(1, 5), 'a'
UNION ALL
SELECT generate_series(6, 10), 'b';
```

Komutlarını çalıştırarak ilgili tabloyu oluşturup, istenen verileri yükledik. Sonrasında;

a1-)

```
-- H1 hareketi: 'b' olan satırları 'a' yapma
BEGIN;
UPDATE T SET name = 'a' WHERE name = 'b';

-- H2 hareketi: 'a' olan satırları 'b' yapma
UPDATE T SET name = 'b' WHERE name = 'a';

-- İşlemi tamamla
COMMIT;
```

<H1,H2> işlemlerini sırayla çalıştırdık ve tablonun son durumu aşağıdaki gibi oldu.

	id [PK] integer	name text
1	1	b
2	2	b
3	3	b
4	4	b
5	5	b
6	6	b
7	7	b
8	8	b
9	9	b
10	10	b

a2-)

Sonrasında;

```
-- H2 hareketi: 'a' olan satırları 'b' yapma
UPDATE T SET name = 'b' WHERE name = 'a';
-- H1 hareketi: 'b' olan satırları 'a' yapma
BEGIN;
UPDATE T SET name = 'a' WHERE name = 'b';
-- İşlemi tamamla
COMMIT;
```

<H2,H1> işlemlerini sırayla çalıştırdık ve tablonun son durumu aşağıdaki gibi oldu.

	id [PK] integer	name text
1	1	a
2	2	a
3	3	a
4	4	a
5	5	a
6	6	a
7	7	a
8	8	a
9	9	a
10	10	a

Bu çıktıları almamızın nedeni, <H1,H2> sırasıyla işlemleri çalıştırırken, ilk olarak H1 hareketi çalıştığı için 'b' olan satırlar **'a'** yapıldı. Ardından H2 hareketi çalıştı ve 'a' olan satırlar da **'b'** yapıldı. Bundan dolayı tüm satırların name'i **'b'** olmuş oldu. <H2,H1> sırasıyla işlemleri çalıştırırken, ilk olarak H2 hareketi çalıştığı için **'a'** olan satırlar **'b'** yapıldı. Ardından H1 hareketi çalıştı ve **'b'** olan satırlar da **'a'** yapıldı. Bundan dolayı tüm satırların name'i **'a'** olmuş oldu.

b1-)

H1 işlemini commit etmeden çalıştırmak için;

```
-- H1 hareketi: 'b' olan satırları 'a' yapma
BEGIN;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
UPDATE T SET name = 'a' WHERE name = 'b';
```

Komutunu kullandık.

b2-)

H2 işlemini commit etmeden çalıştırmak için;

```
-- H2 hareketi: 'a' olan satırları 'b' yapma
BEGIN;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
UPDATE T SET name = 'b' WHERE name = 'a';
```

Komutunu kullandık. Sonrasında her iki işlemi de commit ettikten sonra tablomuzun son hali;

	id [PK] integer	name text
1	1	b
2	2	b
3	3	b
4	4	b
5	5	b
6	6	a
7	7	a
8	8	a
9	9	a
10	10	a

Şeklinde oldu. Bunun sebebi, **REPEATABLE READ** yalıtım seviyesinde, bir işlem (transaction) içinde okunan veriler sabitlenir ve diğer işlemler tarafından değiştirilemez. Her iki hareketi de aynı yalıtım seviyesiyle çalıştırdığınızda, işlemler birbirini engellemez ve her iki hareket de tamamlanır. Sonrasında, her iki hareketi de sıralaması önemsiz şekilde commit ettiğimizde, değişiklikler kalıcı hale gelmiş olur.

c1-)

H1 işlemini commit etmeden **SERIALIZABLE** yalıtım seviyesinde çalıştırmak için;

```
-- H1 hareketi: 'b' olan satırları 'a' yapma
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
UPDATE T SET name = 'a' WHERE name = 'b';
-- COMMIT; -- H1 hareketini commit etmeyin
```

Komutunu kullandık.

c2-)

H2 işlemini commit etmeden **SERIALIZABLE** yalıtım seviyesinde çalıştırmak için;

```
-- H2 hareketi: 'a' olan satırları 'b' yapma
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
UPDATE T SET name = 'b' WHERE name = 'a';
-- COMMIT; -- H2 hareketini commit etmeyin
```

Komutunu kullandık. Sonrasında önce H1 işlemini commit ettik. Sonra H2 işlemini commit etmeye çalışırken

```
ERROR:  could not serialize access due to read/write dependencies among transactions
DETAIL:  Reason code: Canceled on identification as a pivot, during commit attempt.
HINT:  The transaction might succeed if retried.
SQL state: 40001
```

Hatasını alıyoruz. Bu hata **SERIALIZABLE** yalıtım seviyesinde iki işlem arasında çakışmanın oluştuğunu gösterir. Bu hatanın sebebi, H1 hareketinin commit edildikten sonra H2 hareketinin başlamadan önce, H1 hareketinin yaptığı değişikliklere bağımlı olmasıdır.

Sonra tablomuzun son hali;

	id [PK] integer	name text
1	1	a
2	2	a
3	3	a
4	4	a
5	5	a
6	6	a
7	7	a
8	8	a
9	9	a
10	10	a

Şeklinde oldu. Bunun sebebi, **SERIALIZABLE** yalıtım seviyesinde, işlemler sırayla çalıştırılır ve işlemler arasında çakışma engellenir. H1 hareketi başladıktan sonra H2 hareketi çalıştırılmış olsa bile, H1 hareketi commit edildikten sonra H2 hareketi commit edilmeye çalışılırken geri çevrildiği için, H2 hareketi herhangi bir değişikliğe sebep olmaz. H1 hareketi, **'b'** olan satırları **'a'** yapar. Ancak, H2 hareketi H1 hareketinin yaptığı değişikliklere bağımlı olduğu için commit işlemi sırasında çakışma oluşur ve PostgreSQL otomatik olarak H2 hareketini geri çevirir.

Aynı işlemleri, commit sırasını değiştirerek yaptığımızda;

	id [PK] integer	name text
1	1	b
2	2	b
3	3	b
4	4	b
5	5	b
6	6	b
7	7	b
8	8	b
9	9	b
10	10	b

Çıktısını aldık. Bunun sebebi, yine **SERIALIZABLE** yalıtım seviyesinde, işlemlerin sırayla çalıştırılması ve çakışmaların engellenmesidir. H2 hareketi başladıktan sonra H1 hareketi çalıştırılmış olsa bile, H2 hareketi commit edildiğinde H1 hareketi geri çevrilmiştir ve bundan dolayı H1 hareketi herhangi bir değişikliğe sebep olmamıştır. H2 hareketi, **'a'** olan satırları **'b'** yapmıştır ve ilk olarak bu hareketi commit ettiğimiz için ve H1 hareketi H2 hareketinin yaptığı değişikliklere bağımlı olduğu için H1'nin commit işlemi sırasında çakışma oluşur ve PostgreSQL otomatik olarak H1 hareketini geri çevirir.