IF3140 MANAJEMEN BASIS DATA

MEKANISME CONCURRENCY CONTROL DAN RECOVERY

K02 Kelompok 05

Anggota :

Ghazi Akmal Fauzan	13521058
Ilham Akbar	13521068
Althaaf Khasyi Atisomya	13521130
Johanes Lee	13521148

Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2023

Daftar Isi

1. Eksplorasi Transaction Isolation

a. Serializable

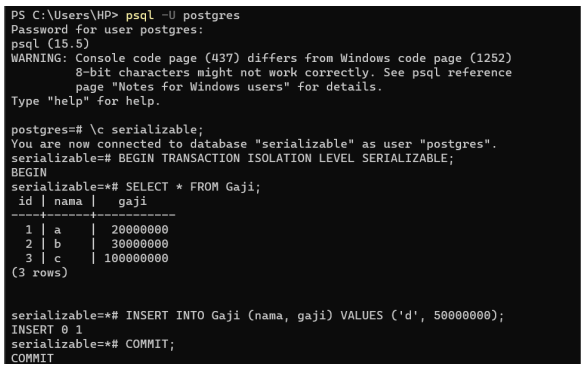
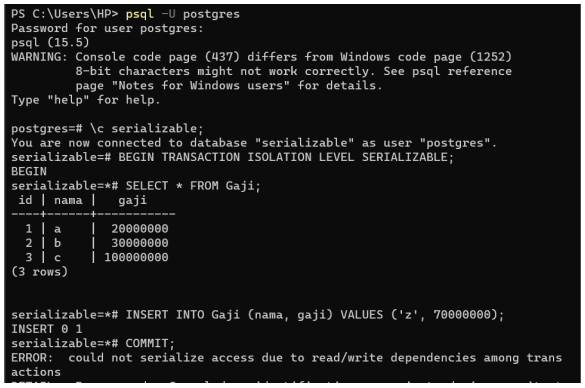
Derajat isolasi pada PostgreSQL mengacu pada interaksi antara transaksi dalam basis data. Terdapat 4 tingkat derajat isolasi yaitu serializable, repeatable read, read committed, dan read uncommitted. Serializable merupakan derajat isolasi dengan isolasi paling tinggi dan ketat. Isolasi ini menjamin bahwa eksekusi bersamaan dari beberapa transaksi serializable akan menghasilkan hasil yang sama dengan menjalankannya satu per satu dalam beberapa urutan. Artinya setiap transaksi serializable akan melihat database dalam keadaan yang konsisten dan bebas dari fenomena aneh yang disebabkan oleh interaksi dengan transaksi lain seperti dirty read, non repeatable read, phantom read, dan serialization anomaly. Namun, aplikasi yang menggunakan isolasi ini harus siap untuk mengulangi transaksi karena kegagalan serialisasi.

Serializable bekerja hampir sama seperti Repeatable Read. Namun, ada perbedaan penting dalam cara level isolasi Serializable memperlakukan transaksi dan memantau kondisi yang dapat menyebabkan eksekusi bersama dari sekelompok transaksi serializable berperilaku secara tidak konsisten dengan semua eksekusi serial yang mungkin. Dalam level isolasi Serializable tidak menambahkan pemblokiran tambahan dibandingkan dengan level isolasi Repeatable Read. Jika sistem basis data mendeteksi bahwa ada kondisi yang dapat menyebabkan serialization anomaly, maka sistem basis data akan menyebabkan kegagalan serialisasi.

Untuk mencapai serialisabilitas yang sebenarnya, PostgreSQL menggunakan kunci predikat (predicate locking). Kunci predikat ini menyimpan informasi tentang ketergantungan antara transaksi-seriabel bersamaan. Kunci ini tidak menyebabkan pemblokiran atau deadlock karena digunakan untuk mengidentifikasi dan menandai ketergantungan antara transaksi serializable yang dapat menyebabkan serialization anomaly.

Berikut merupakan simulasi dari isolasi serializable.

No	Terminal	Query

1.		BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE; SELECT * FROM Gaji; INSERT INTO Gaji (nama, gaji) VALUES ('d', 50000000); COMMIT;
2.		BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE; SELECT * FROM Gaji; INSERT INTO Gaji (nama, gaji) VALUES ('z', 70000000); COMMIT;
Penjelasan		
Setelah menjalankan kedua transaksi, perhatikan bahwa salah satu transaksi akan berhasil melakukan commit, sementara yang lain akan di-rollback. Hal ini terjadi		

karena hasil yang diharapkan dari kedua transaksi tidak konsisten dengan urutan serial yang mungkin. Dalam contoh ini, jika transaksi A dijalankan sebelum transaksi B, hasilnya akan berbeda. Dengan demikian, tingkat isolasi Serializable mencegah hasil yang tidak konsisten dan memastikan bahwa transaksi berjalan seperti yang diharapkan dalam urutan serial yang dimungkinkan.

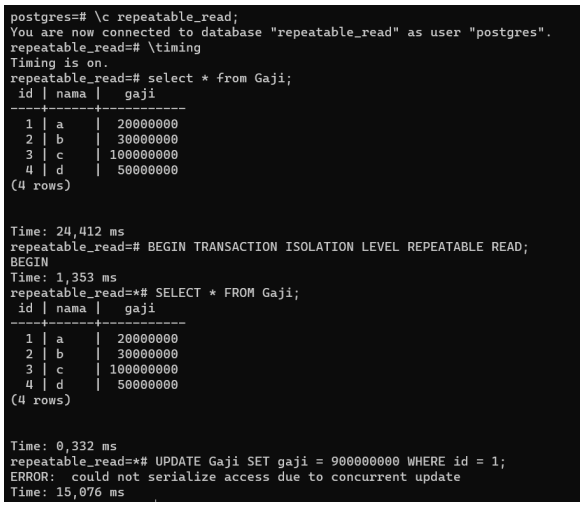
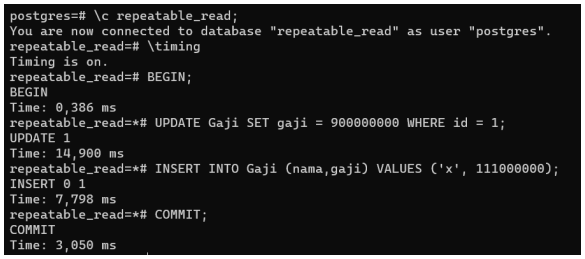
b. Repeatable Read

Tingkat isolasi Repeatable read merupakan salah satu cara untuk menjaga konsistensi dari transaksi dalam basis data. Repeatable read menjamin bahwa semua transaksi dalam basis data akan melihat snapshot data yang konsisten sepanjang durasi transaksi tersebut. Transaksi pada tingkat ini tidak akan melihat data yang belum di-commit atau perubahan yang di-commit oleh transaksi konkuren selama eksekusi transaksi tersebut. Namun, setiap query akan melihat hasil dari update sebelumnya yang dilakukan dalam transaksi itu sendiri, meskipun update tersebut belum di-commit.

Repeatable read memiliki kemampuan untuk memberikan fasilitas mekanisme tanpa pemblokiran. Hal ini berarti transaksi mungkin untuk membaca data tanpa menghentikan transaksi lainnya, sehingga menciptakan efisiensi dalam operasi basis data. Mekanisme tanpa pemblokiran ini memungkinkan transaksi untuk bekerja secara independen dan mengakses data yang konsisten, tanpa harus menunggu atau mengganggu transaksi lain yang sedang berjalan. Dengan demikian, Repeatable Read mampu mengoptimalkan kinerja dan throughput dalam lingkungan basis data yang padat dan sibuk.

Repeatable Read juga berhasil menangani fenomena aneh seperti phantom read dengan memastikan bahwa setiap baris data yang dibaca tetap terlihat dan tidak muncul atau hilang selama transaksi berlangsung. Namun, perlu diingat bahwa aplikasi yang mengadopsi tingkat isolasi ini harus siap dengan potensi rollback dalam menghadapi konflik.

Berikut merupakan simulasi dari isolasi repeatable read.

No	Terminal	Query
1.		BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ; SELECT * FROM Gaji; UPDATE Gaji SET gaji = 900000000 WHERE id = 1;
2.		BEGIN; UPDATE Gaji SET gaji = 900000000 WHERE id = 1; INSERT INTO Gaji (nama,gaji) VALUES ('x', 111000000); COMMIT;
Penjelasan		
Perintah "UPDATE Gaji SET gaji = 900000000 WHERE id = 1;" gagal dalam simulasi transaksi repeatable read karena adanya konflik dengan transaksi sebelumnya yang belum di-commit. Dalam level isolasi repeatable read, ketika		

transaksi baru dimulai, semua data yang akan diakses akan dikunci dan tidak dapat diubah oleh transaksi lain hingga transaksi saat ini selesai (commit atau rollback).

Dalam contoh ini, pada transaksi sebelumnya, terdapat perintah "UPDATE Gaji SET gaji = 900000000 WHERE id = 1;" yang mengubah nilai gaji menjadi 900.000.000 pada baris dengan id 1 di tabel Gaji. Namun, perubahan ini belum di-commit.

Ketika transaksi baru dengan level isolasi repeatable read dimulai, perintah "SELECT * FROM Gaji;" dijalankan untuk mengambil semua data dari tabel Gaji. Perintah ini mengembalikan hasil yang sesuai dengan kondisi data pada saat transaksi dimulai. Namun, karena perubahan pada transaksi sebelumnya belum di-commit, perintah "UPDATE Gaji SET gaji = 900000000 WHERE id = 1;" pada transaksi baru akan terblokir dan menunggu hingga transaksi sebelumnya dilakukan commit atau rollback.

Dengan demikian, perintah "UPDATE Gaji SET gaji = 900000000 WHERE id = 1;" pada transaksi baru gagal karena terjadi konflik dengan perubahan yang belum di-commit oleh transaksi sebelumnya.

c. Read Committed

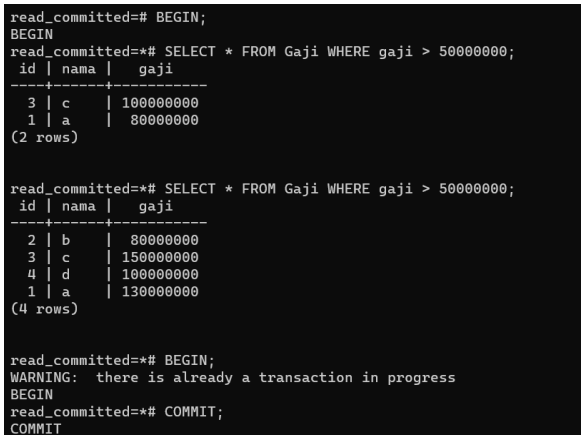
Tingkat isolasi read committed merupakan tingkat isolasi default pada PostgreSQL. Tingkat isolasi ini dirancang untuk mencapai keseimbangan antara konsistensi data dan efisiensi pengelolaan transaksi. Dalam konteks ini, Read Committed menyediakan gambaran yang konsisten dari data yang telah di-commit "Consistent Snapshot". Artinya, jika suatu transaksi telah mengubah atau memasukkan data ke dalam basis data dan telah di-commit, transaksi lain yang menjalankan operasi pembacaan (read) akan melihat perubahan data tersebut. Dengan kata lain, perubahan data yang terjadi dalam transaksi yang telah di-commit akan terlihat oleh transaksi lainnya.

Namun, Read Committed juga memungkinkan transaksi lain untuk mengakses data tersebut tanpa pembatasan yang berlebihan. Ini berarti bahwa transaksi lain dapat membaca data yang telah di-commit tanpa harus menunggu hingga transaksi tersebut

selesai sepenuhnya. Hal ini memberikan fleksibilitas dan efisiensi dalam mengakses data, karena tidak ada pembatasan yang ketat terhadap transaksi pembacaan. Meskipun tidak menawarkan isolasi setinggi tingkat Serializable atau Repeatable Read, tingkat isolasi ini tetap memberikan perlindungan terhadap fenomena aneh seperti non-repeatable read dan phantom read.

Dalam keseluruhan, tingkat isolasi Read Committed dalam PostgreSQL memberikan keseimbangan yang baik antara konsistensi data dan efisiensi transaksi. Data yang telah di-commit dapat dilihat oleh transaksi lainnya, sehingga memastikan gambaran yang konsisten dari basis data.

Berikut merupakan simulasi dari isolasi read committed.

<table>
<tr><th>No</th><th>Terminal</th><th>Query</th></tr>
<tr><td>1.</td><td> 
</td><td>BEGIN;

SELECT * FROM Gaji WHERE gaji > 50000000;

SELECT * FROM Gaji WHERE gaji > 50000000;

COMMIT;</td></tr>
<tr><td>2.</td><td> 
</td><td>BEGIN;

UPDATE Gaji SET gaji = gaji + 50000000;

COMMIT;</td></tr>
<tr><td colspan="3">Penjelasan</td></tr>
<tr><td colspan="3">Pada contoh di atas, dalam transaksi pertama, SELECT query pertama hanya akan melihat data yang telah ter-commit sebelum transaksi tersebut dimulai. Oleh karena itu, jika UPDATE query pada Transaksi 2 belum ter-commit, perubahan yang dilakukan oleh UPDATE query tersebut tidak akan terlihat oleh SELECT query pada Transaksi 1.</td></tr>
</table>

d. Read Uncommitted

Read Uncommitted adalah salah satu tingkat isolasi transaksi dalam sistem manajemen basis data. Tingkat isolasi ini memungkinkan transaksi untuk membaca perubahan yang dilakukan oleh transaksi lain sebelum perubahan tersebut dikonfirmasi (committed). Dalam Read Uncommitted, tidak ada penguncian (locking) yang diterapkan pada data yang sedang dibaca, sehingga transaksi lain dapat mengubah data tersebut sebelum transaksi awal menyelesaikan operasinya. Transaksi tidak diharuskan untuk menunggu sampai transaksi lain selesai atau mengunci data yang sedang dibaca. Ini berarti bahwa transaksi dapat melihat perubahan yang belum dikonfirmasi (dirty read), yang dapat menyebabkan inkonsistensi jika perubahan tersebut akhirnya dibatalkan atau di-rollback.

Tingkat isolasi read uncommitted dalam PostgreSQL memiliki perilaku seperti read committed. Hal ini disebabkan oleh Multiversion Concurrency Control (MVCC). Dalam MVCC, setiap transaksi memiliki tampilan yang konsisten dan terisolasi dari data yang berubah oleh transaksi lain. Dalam tingkat isolasi Read Committed, transaksi hanya dapat melihat data yang telah dikonfirmasi (committed) oleh transaksi lain. Data yang sedang dalam proses perubahan oleh transaksi lain tidak akan terlihat oleh transaksi yang berjalan pada tingkat ini.

Read Uncommitted memberikan tingkat isolasi yang lebih rendah dibandingkan dengan tingkat isolasi lainnya seperti Serializable, Repeatable Read, dan Read Committed. Oleh karena itu, pada skenario di mana kecepatan akses data langsung lebih diutamakan daripada konsistensi tingkat tinggi maka read uncommitted ini akan dipilih. Meskipun menawarkan fleksibilitas dalam operasi waktu nyata, Read Uncommitted harus memiliki potensi risiko yang terkait dengan akses data yang belum di-commit.

2. Implementasi Concurrency Control Protocol

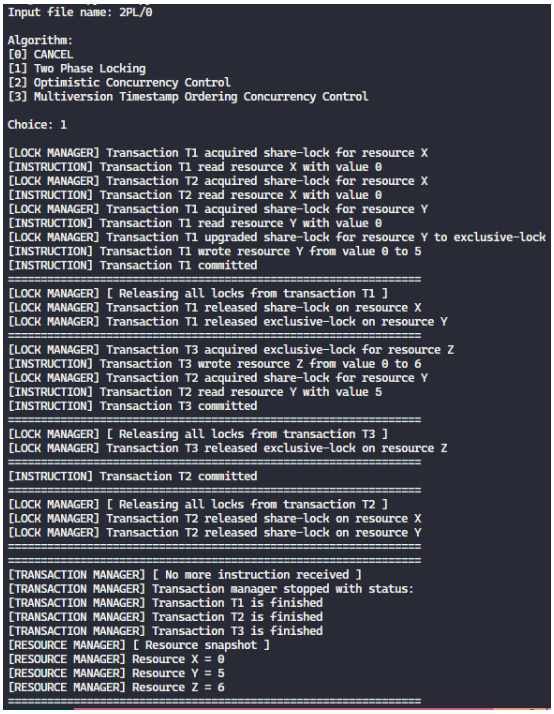
a. Two-Phase Locking (2PL)

Nama Kelas	**Deskripsi**
WriteInstructionWithLock	Kelas instruksi yang berisi eksekusi instruksi *write* untuk algoritma 2PL
ReadInstructionWithLock	Kelas instruksi yang berisi eksekusi instruksi *read* untuk algoritma 2PL
CommitInstructionWithLock	Kelas instruksi yang berisi eksekusi instruksi *commit* untuk algoritma 2PL
LockManager	Manager yang mengatur akuisisi dan pelepasan *lock* terhadap *resource*
TwoPhaseResourceHandler	Kelas yang mengatur akses *resource* yang ada, termasuk *rollback* untuk *resource* yang pernah ditulis
TwoPhaseTransactionManager	Kelas yang mengatur eksekusi instruksi oleh berbagai transaksi yang ada
TwoPhaseTransaction	Kelas yang menyimpan informasi terkait transaksi pernah yang diterima instruksinya oleh TwoPhaseTransactionManager
TwoPhaseInstructionReader	Kelas yang membaca *file* masukan dan memberikan objek instruksi (berdasarkan setiap baris yang dibaca pada *file* tersebut) untuk dieksekusi oleh TwoPhaseTransactionManager

Berikut merupakan simulasi dari Two-Phase Locking Concurrency Control

Input	Output

```
# 2PL Case:
# Simple schedule without waiting and rollback

R T1 X
R T2 X
R T1 Y
W T1 Y=5
C T1
W T3 Z=6
R T2 Y
C T3
C T2
```



Penjelasan:

Simulasi ini menerapkan protokol penguncian 2PL untuk mengelola transaksi dalam sistem basis data. Transaksi T1 membaca dan mengubah nilai sumber daya X dan Y, dengan kunci yang dibebaskan setelah commit. Transaksi T3 melakukan operasi serupa pada sumber daya Z. Transaksi T2 membaca nilai yang telah diubah oleh T1. Setelah semua transaksi selesai, kunci dilepaskan, dan program memberikan snapshot akhir dari nilai-nilai sumber daya (X = 0, Y = 5, Z = 6). Pesan log menyediakan rincian mengenai pengelolaan kunci, instruksi transaksi, dan status transaksi, memberikan pemahaman menyeluruh tentang proses pengelolaan transaksi.

```
# 2PL Case:
# Involve wait-queue

R T1 X
W T2 X=5
C T1
C T2
```

```
Input file name: 2PL/1

Algorithm:
[0] CANCEL
[1] Two Phase Locking
[2] Optimistic Concurrency Control
[3] Multiversion Timestamp Ordering Concurrency Control

Choice: 1

[LOCK MANAGER] Transaction T1 acquired share-lock for resource X
[INSTRUCTION] Transaction T1 read resource X with value 0
[TRANSACTION MANAGER] Instruction W(X=5) from transaction T2 entered wait-queue
[INSTRUCTION] Transaction T1 committed
============================================================
[LOCK MANAGER] [ Releasing all locks from transaction T1 ]
[LOCK MANAGER] Transaction T1 released share-lock on resource X
============================================================
[TRANSACTION MANAGER] Instruction W(X=5) from transaction T2 leave wait-queue
[LOCK MANAGER] Transaction T2 acquired exclusive-lock for resource X
[INSTRUCTION] Transaction T2 wrote resource X from value 0 to 5
[INSTRUCTION] Transaction T2 committed
============================================================
[LOCK MANAGER] [ Releasing all locks from transaction T2 ]
[LOCK MANAGER] Transaction T2 released exclusive-lock on resource X
============================================================
============================================================
[TRANSACTION MANAGER] [ No more instruction received ]
[TRANSACTION MANAGER] Transaction manager stopped with status:
[TRANSACTION MANAGER] Transaction T1 is finished
[TRANSACTION MANAGER] Transaction T2 is finished
[RESOURCE MANAGER] [ Resource snapshot ]
[RESOURCE MANAGER] Resource X = 5
============================================================
```

Penjelasan:

Simulasi ini mengilustrasikan penggunaan protokol 2PL dalam sistem basis data dengan antrian tunggu (wait-queue) saat ada penguncian yang terlibat. Transaksi T1 membaca sumber daya X dan kemudian commit. Transaksi T2 mencoba menulis nilai 5 ke sumber daya X, tetapi karena T1 masih memegang kunci, T2 memasuki antrian tunggu. Setelah T1 selesai dan melepaskan kunci, T2 meninggalkan antrian tunggu, mengakuisisi kunci, menulis nilai baru, dan commit. Output mencerminkan langkah-langkah ini, termasuk pesan mengenai masuk dan keluar dari antrian tunggu serta pembebasan kunci setelah setiap transaksi selesai. Akhirnya, program menampilkan snapshot akhir dari nilai sumber daya X (X = 5).

```
# 2PL Case:
# Involve rollback and wait-queue

R T1 X
R T2 Y
R T2 X
W T1 Y=5
W T2 X=5
C T2
C T1
```

```
Input file name: 2PL/2

Algorithm:
[0] CANCEL
[1] Two Phase Locking
[2] Optimistic Concurrency Control
[3] Multiversion Timestamp Ordering Concurrency Control

Choice: 1

[LOCK MANAGER] Transaction T1 acquired share-lock for resource X
[INSTRUCTION] Transaction T1 read resource X with value 0
[LOCK MANAGER] Transaction T2 acquired share-lock for resource Y
[INSTRUCTION] Transaction T2 read resource Y with value 0
[LOCK MANAGER] Transaction T2 acquired share-lock for resource X
[INSTRUCTION] Transaction T2 read resource X with value 0
[TRANSACTION MANAGER] Transaction T2 is aborting
============================================================
[RESOURCE MANAGER] [ Rolling back resource values from updates of transaction T2 ]
[RESOURCE MANAGER] Transaction T2 has not updated any resource
============================================================
============================================================
[LOCK MANAGER] [ Releasing all locks from transaction T2 ]
[LOCK MANAGER] Transaction T2 released share-lock on resource Y
[LOCK MANAGER] Transaction T2 released share-lock on resource X
============================================================
[LOCK MANAGER] Transaction T1 acquired exclusive-lock for resource Y
[INSTRUCTION] Transaction T1 wrote resource Y from value 0 to 5
[TRANSACTION MANAGER] Trying to rollback transaction T2
[TRANSACTION MANAGER] Instruction R(Y) from transaction T2 entered wait-queue
[TRANSACTION MANAGER] [ Transaction T2 is waiting for previous instructions ]
[TRANSACTION MANAGER] Instruction R(X) from transaction T2 entered wait-queue
[TRANSACTION MANAGER] [ Transaction T2 is waiting for previous instructions ]
[TRANSACTION MANAGER] Instruction W(X=5) from transaction T2 entered wait-queue
[TRANSACTION MANAGER] [ Transaction T2 is waiting for previous instructions ]
[TRANSACTION MANAGER] Instruction commit from transaction T2 entered wait-queue
[INSTRUCTION] Transaction T1 committed
```

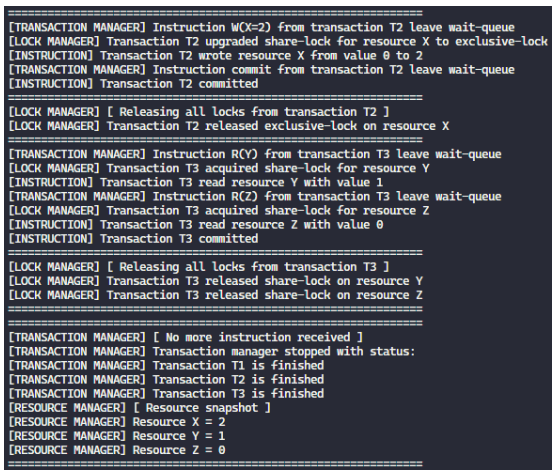
```
[LOCK MANAGER] [ Releasing all locks from transaction T1 ]
[LOCK MANAGER] Transaction T1 released share-lock on resource X
[LOCK MANAGER] Transaction T1 released exclusive-lock on resource Y
[TRANSACTION MANAGER] Instruction R(Y) from transaction T2 leave wait-queue
[LOCK MANAGER] Transaction T2 acquired share-lock for resource Y
[INSTRUCTION] Transaction T2 read resource Y with value 5
[TRANSACTION MANAGER] Instruction R(X) from transaction T2 leave wait-queue
[LOCK MANAGER] Transaction T2 acquired share-lock for resource X
[INSTRUCTION] Transaction T2 read resource X with value 0
[TRANSACTION MANAGER] Instruction W(X=5) from transaction T2 leave wait-queue
[LOCK MANAGER] Transaction T2 upgraded share-lock for resource X to exclusive-lock
[INSTRUCTION] Transaction T2 wrote resource X from value 0 to 5
[TRANSACTION MANAGER] Instruction commit from transaction T2 leave wait-queue
[INSTRUCTION] Transaction T2 committed
[LOCK MANAGER] [ Releasing all locks from transaction T2 ]
[LOCK MANAGER] Transaction T2 released share-lock on resource Y
[LOCK MANAGER] Transaction T2 released exclusive-lock on resource X
[TRANSACTION MANAGER] [ No more instruction received ]
[TRANSACTION MANAGER] Transaction manager stopped with status:
[TRANSACTION MANAGER] Transaction T1 is finished
[TRANSACTION MANAGER] Transaction T2 is finished
[RESOURCE MANAGER] [ Resource snapshot ]
[RESOURCE MANAGER] Resource X = 5
[RESOURCE MANAGER] Resource Y = 5
```

Penjelasan:

Simulasi ini menunjukkan implementasi protokol 2PL dalam sistem basis data yang melibatkan rollback dan antrian tunggu. Transaksi T1 membaca sumber daya X dan kemudian menulis nilai 5 ke sumber daya Y. Transaksi T2 membaca sumber daya Y dan X, tetapi karena T2 harus di-rollback, semua instruksi T2 masuk ke dalam antrian tunggu. Setelah T1 selesai, T2 melepaskan kunci dan mulai proses rollback. Setelah berhasil melakukan rollback, T2 menunggu instruksi sebelumnya dan kemudian menyelesaikan operasi, termasuk membaca dan menulis sumber daya Y dan X. Akhirnya, program menampilkan snapshot akhir dari nilai sumber daya X dan Y (X = 5, Y = 5), serta status transaksi yang telah selesai (T1 dan T2).

```
# 2PL Case:
# Involve wait-queue and rollback
# Rolling back when there are instructions in wait-queue for the same transaction
# Fail when rolling back will make all rollback instructions enter wait-queue

R T1 X
R T2 X
R T3 Y
R T1 Y
W T2 X=2
C T2
R T3 Z
W T1 Y=1
C T1
C T3
```

```
Input file name: 2PL/3

Algorithm:
[0] CANCEL
[1] Two Phase Locking
[2] Optimistic Concurrency Control
[3] Multiversion Timestamp Ordering Concurrency Control

Choice: 1

[LOCK MANAGER] Transaction T1 acquired share-lock for resource X
[INSTRUCTION] Transaction T1 read resource X with value 0
[LOCK MANAGER] Transaction T2 acquired share-lock for resource X
[INSTRUCTION] Transaction T2 read resource X with value 0
[LOCK MANAGER] Transaction T3 acquired share-lock for resource Y
[INSTRUCTION] Transaction T3 read resource Y with value 0
[LOCK MANAGER] Transaction T1 acquired share-lock for resource Y
[INSTRUCTION] Transaction T1 read resource Y with value 0
[TRANSACTION MANAGER] Instruction W(X=2) from transaction T2 entered wait-queue
[TRANSACTION MANAGER] [ Transaction T2 is waiting for previous instructions ]
[TRANSACTION MANAGER] Instruction commit from transaction T2 entered wait-queue
[LOCK MANAGER] Transaction T3 acquired share-lock for resource Z
[INSTRUCTION] Transaction T3 read resource Z with value 0
[TRANSACTION MANAGER] Transaction T3 is aborting
[RESOURCE MANAGER] [ Rolling back resource values from updates of transaction T3 ]
[RESOURCE MANAGER] Transaction T3 has not updated any resource
[LOCK MANAGER] [ Releasing all locks from transaction T3 ]
[LOCK MANAGER] Transaction T3 released share-lock on resource Y
[LOCK MANAGER] Transaction T3 released share-lock on resource Z
[LOCK MANAGER] Transaction T1 upgraded share-lock for resource Y to exclusive-lock
[INSTRUCTION] Transaction T1 wrote resource Y from value 0 to 1
[TRANSACTION MANAGER] Trying to rollback transaction T3
[TRANSACTION MANAGER] Instruction R(Y) from transaction T3 entered wait-queue
[TRANSACTION MANAGER] [ Transaction T3 is waiting for previous instructions ]
[TRANSACTION MANAGER] Instruction R(Z) from transaction T3 entered wait-queue
[INSTRUCTION] Transaction T1 committed
[LOCK MANAGER] [ Releasing all locks from transaction T1 ]
[LOCK MANAGER] Transaction T1 released share-lock on resource X
[LOCK MANAGER] Transaction T1 released exclusive-lock on resource Y
```



Penjelasan:

Simulasi ini menggambarkan implementasi protokol 2PL dalam sistem basis data dengan antrian tunggu dan rollback, khususnya ketika terjadi kegagalan rollback karena semua instruksi rollback akan masuk ke antrian tunggu. Transaksi T1 membaca dan mengubah nilai X dan Y, T2 membaca dan menulis nilai X, dan T3 membaca nilai Y dan Z. Setelah T2 di-rollback, semua instruksi rollback T3 masuk antrian tunggu. Karena tidak mungkin melakukan rollback T3 tanpa menunggu instruksi sebelumnya, program menghentikan proses dan menunjukkan bahwa transaksi T3 gagal. Meskipun T2 dan T3 gagal, T1 berhasil menyelesaikan dan melepaskan kunci. Snapshot akhir menunjukkan nilai sumber daya (X = 2, Y = 1, Z = 0), dan status transaksi ditampilkan sebagai selesai atau gagal sesuai dengan hasil eksekusi.

```
# 2PL Case:
# Involve wait-queue and rollback
# Involve rolling back written resource
# Rolling back when there are instructions in wait-queue for the same transaction
# Fail when rolling back will make all rollback instructions enter wait-queue

R T1 Z
R T2 X
R T3 Y
W T3 Y=10
R T1 Y
W T2 X=2
C T2
R T3 Z
W T1 Y=1
C T1
C T3
```

```
Input file name: 2PL/4

Algorithm:
[0] CANCEL
[1] Two Phase Locking
[2] Optimistic Concurrency Control
[3] Multiversion Timestamp Ordering Concurrency Control

Choice: 1

[LOCK MANAGER] Transaction T1 acquired share-lock for resource Z
[INSTRUCTION] Transaction T1 read resource Z with value 0
[LOCK MANAGER] Transaction T2 acquired share-lock for resource X
[INSTRUCTION] Transaction T2 read resource X with value 0
[LOCK MANAGER] Transaction T3 acquired share-lock for resource Y
[INSTRUCTION] Transaction T3 read resource Y with value 0
[LOCK MANAGER] Transaction T3 upgraded share-lock for resource Y to exclusive-lock
[INSTRUCTION] Transaction T3 wrote resource Y from value 0 to 10
[TRANSACTION MANAGER] Transaction T3 is aborting
============================================================
[RESOURCE MANAGER] [ Rolling back resource values from updates of transaction T3 ]
[RESOURCE MANAGER] Wrote back resource Y from 10 to 0
============================================================
============================================================
[LOCK MANAGER] [ Releasing all locks from transaction T3 ]
[LOCK MANAGER] Transaction T3 released exclusive-lock on resource Y
============================================================
[LOCK MANAGER] Transaction T1 acquired share-lock for resource Y
[INSTRUCTION] Transaction T1 read resource Y with value 0
[TRANSACTION MANAGER] Trying to rollback transaction T3
[LOCK MANAGER] Transaction T3 acquired share-lock for resource Y
[INSTRUCTION] Transaction T3 read resource Y with value 0
[TRANSACTION MANAGER] Instruction W(Y=10) from transaction T3 entered wait-queue
[LOCK MANAGER] Transaction T2 upgraded share-lock for resource X to exclusive-lock
[INSTRUCTION] Transaction T2 wrote resource X from value 0 to 2
[INSTRUCTION] Transaction T2 committed
============================================================
[LOCK MANAGER] [ Releasing all locks from transaction T2 ]
[LOCK MANAGER] Transaction T2 released exclusive-lock on resource X
============================================================
[TRANSACTION MANAGER] Instruction W(Y=10) from transaction T3 leave wait-queue
[TRANSACTION MANAGER] Instruction W(Y=10) from transaction T3 entered wait-queue
[TRANSACTION MANAGER] [ Transaction T3 is waiting for previous instructions ]
[TRANSACTION MANAGER] Instruction R(Z) from transaction T3 entered wait-queue
[TRANSACTION MANAGER] Transaction T3 is aborting
============================================================
[RESOURCE MANAGER] [ Rolling back resource values from updates of transaction T3 ]
[RESOURCE MANAGER] Transaction T3 has not updated any resource
```

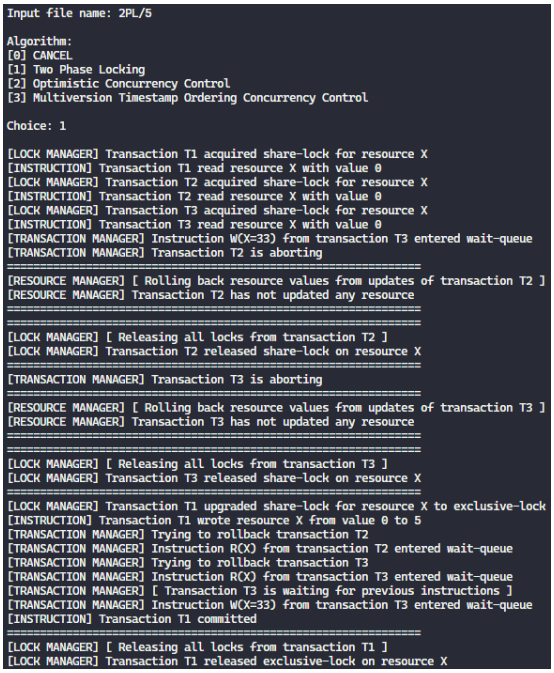
```
============================================================
[LOCK MANAGER] [ Releasing all locks from transaction T3 ]
[LOCK MANAGER] Transaction T3 released share-lock on resource Y
============================================================
[LOCK MANAGER] Transaction T1 upgraded share-lock for resource Y to exclusive-lock
[INSTRUCTION] Transaction T1 wrote resource Y from value 0 to 1
[TRANSACTION MANAGER] Trying to rollback transaction T3
[TRANSACTION MANAGER] Instruction R(Y) from transaction T3 entered wait-queue
[TRANSACTION MANAGER] [ Transaction T3 is waiting for previous instructions ]
[TRANSACTION MANAGER] Instruction W(Y=10) from transaction T3 entered wait-queue
[TRANSACTION MANAGER] [ Transaction T3 is waiting for previous instructions ]
[TRANSACTION MANAGER] Instruction R(Z) from transaction T3 entered wait-queue
[INSTRUCTION] Transaction T1 committed
============================================================
[LOCK MANAGER] [ Releasing all locks from transaction T1 ]
[LOCK MANAGER] Transaction T1 released share-lock on resource Z
[LOCK MANAGER] Transaction T1 released exclusive-lock on resource Y
============================================================
[TRANSACTION MANAGER] Instruction R(Y) from transaction T3 leave wait-queue
[LOCK MANAGER] Transaction T3 acquired share-lock for resource Y
[INSTRUCTION] Transaction T3 read resource Y with value 1
[TRANSACTION MANAGER] Instruction W(Y=10) from transaction T3 leave wait-queue
[LOCK MANAGER] Transaction T3 upgraded share-lock for resource Y to exclusive-lock
[INSTRUCTION] Transaction T3 wrote resource Y from value 1 to 10
[TRANSACTION MANAGER] Instruction R(Z) from transaction T3 leave wait-queue
[LOCK MANAGER] Transaction T3 acquired share-lock for resource Z
[INSTRUCTION] Transaction T3 read resource Z with value 0
[INSTRUCTION] Transaction T3 committed
============================================================
[LOCK MANAGER] [ Releasing all locks from transaction T3 ]
[LOCK MANAGER] Transaction T3 released exclusive-lock on resource Y
[LOCK MANAGER] Transaction T3 released share-lock on resource Z
============================================================
============================================================
[TRANSACTION MANAGER] [ No more instruction received ]
[TRANSACTION MANAGER] Transaction manager stopped with status:
[TRANSACTION MANAGER] Transaction T1 is finished
[TRANSACTION MANAGER] Transaction T2 is finished
[TRANSACTION MANAGER] Transaction T3 is finished
[RESOURCE MANAGER] [ Resource snapshot ]
[RESOURCE MANAGER] Resource Z = 0
[RESOURCE MANAGER] Resource X = 2
[RESOURCE MANAGER] Resource Y = 10
============================================================
```

Penjelasan:

Simulasi ini menunjukkan implementasi protokol 2PL dalam sistem basis data dengan antrian tunggu dan rollback, termasuk kasus ketika perlu mengembalikan nilai sumber daya yang telah ditulis. Transaksi T1 membaca sumber daya Z dan Y, T2 membaca dan menulis nilai X, dan T3 menulis nilai 10 ke Y sebelum di-rollback. Setelah T3 di-rollback, nilai Y dikembalikan ke 0. Meskipun T2 telah selesai, T1 kemudian menulis nilai 1 ke Y. Proses rollback T3 diulangi karena ada instruksi tertunda. Setelah itu, T3

menulis nilai 10 ke Y dan membaca nilai Z sebelum selesai. Snapshot akhir menunjukkan nilai sumber daya (Z = 0, X = 2, Y = 10), dan status transaksi ditampilkan sebagai selesai atau gagal sesuai dengan hasil eksekusi. Program juga menunjukkan detail perubahan nilai sumber daya selama proses rollback.

```
# 2PL Case:
# Involve wait-queue and rollback
# Involve more than 1 transaction when rollback

R T1 X
R T2 X
R T3 X
W T3 X=33
W T1 X=5
C T1
C T2
C T3
```



```
Input file name: 2PL/5

Algorithm:
[0] CANCEL
[1] Two Phase Locking
[2] Optimistic Concurrency Control
[3] Multiversion Timestamp Ordering Concurrency Control

Choice: 1

[LOCK MANAGER] Transaction T1 acquired share-lock for resource X
[INSTRUCTION] Transaction T1 read resource X with value 0
[LOCK MANAGER] Transaction T2 acquired share-lock for resource X
[INSTRUCTION] Transaction T2 read resource X with value 0
[LOCK MANAGER] Transaction T3 acquired share-lock for resource X
[INSTRUCTION] Transaction T3 read resource X with value 0
[TRANSACTION MANAGER] Instruction W(X=33) from transaction T3 entered wait-queue
[TRANSACTION MANAGER] Transaction T2 is aborting
============================================================
[RESOURCE MANAGER] [ Rolling back resource values from updates of transaction T2 ]
[RESOURCE MANAGER] Transaction T2 has not updated any resource
============================================================
============================================================
[LOCK MANAGER] [ Releasing all locks from transaction T2 ]
[LOCK MANAGER] Transaction T2 released share-lock on resource X
============================================================
[TRANSACTION MANAGER] Transaction T3 is aborting
============================================================
[RESOURCE MANAGER] [ Rolling back resource values from updates of transaction T3 ]
[RESOURCE MANAGER] Transaction T3 has not updated any resource
============================================================
============================================================
[LOCK MANAGER] [ Releasing all locks from transaction T3 ]
[LOCK MANAGER] Transaction T3 released share-lock on resource X
============================================================
[LOCK MANAGER] Transaction T1 upgraded share-lock for resource X to exclusive-lock
[INSTRUCTION] Transaction T1 wrote resource X from value 0 to 5
[TRANSACTION MANAGER] Trying to rollback transaction T2
[TRANSACTION MANAGER] Instruction R(X) from transaction T2 entered wait-queue
[TRANSACTION MANAGER] Trying to rollback transaction T3
[TRANSACTION MANAGER] Instruction R(X) from transaction T3 entered wait-queue
[TRANSACTION MANAGER] [ Transaction T3 is waiting for previous instructions ]
[TRANSACTION MANAGER] Instruction W(X=33) from transaction T3 entered wait-queue
[INSTRUCTION] Transaction T1 committed
============================================================
[LOCK MANAGER] [ Releasing all locks from transaction T1 ]
[LOCK MANAGER] Transaction T1 released exclusive-lock on resource X
============================================================
[TRANSACTION MANAGER] Instruction R(X) from transaction T2 leave wait-queue
[LOCK MANAGER] Transaction T2 acquired share-lock for resource X
[INSTRUCTION] Transaction T2 read resource X with value 5
[TRANSACTION MANAGER] Instruction R(X) from transaction T3 leave wait-queue
[LOCK MANAGER] Transaction T3 acquired share-lock for resource X
[INSTRUCTION] Transaction T3 read resource X with value 5
[TRANSACTION MANAGER] Instruction W(X=33) from transaction T3 leave wait-queue
[TRANSACTION MANAGER] Instruction W(X=33) from transaction T3 entered wait-queue
[INSTRUCTION] Transaction T2 committed
============================================================
[LOCK MANAGER] [ Releasing all locks from transaction T2 ]
[LOCK MANAGER] Transaction T2 released share-lock on resource X
============================================================
[TRANSACTION MANAGER] Instruction W(X=33) from transaction T3 leave wait-queue
[LOCK MANAGER] Transaction T3 upgraded share-lock for resource X to exclusive-lock
[INSTRUCTION] Transaction T3 wrote resource X from value 5 to 33
[INSTRUCTION] Transaction T3 committed
============================================================
[LOCK MANAGER] [ Releasing all locks from transaction T3 ]
[LOCK MANAGER] Transaction T3 released exclusive-lock on resource X
============================================================
============================================================
[TRANSACTION MANAGER] [ No more instruction received ]
[TRANSACTION MANAGER] Transaction manager stopped with status:
[TRANSACTION MANAGER] Transaction T1 is finished
[TRANSACTION MANAGER] Transaction T2 is finished
[TRANSACTION MANAGER] Transaction T3 is finished
[RESOURCE MANAGER] [ Resource snapshot ]
[RESOURCE MANAGER] Resource X = 33
============================================================
```

Penjelasan:

Simulasi ini menunjukkan implementasi protokol 2PL dalam sistem basis data dengan antrian tunggu dan rollback, melibatkan lebih dari satu transaksi saat proses rollback. Transaksi T1, T2, dan T3 membaca nilai X, tetapi T2 di-rollback tanpa melakukan perubahan. Kemudian, T3 juga di-rollback. Setelah T1 selesai, T2 membaca nilai yang diubah oleh T1 dan menyelesaikan commit. Selanjutnya, T3 melakukan commit

setelah menyelesaikan operasi write yang dijadwalkan sebelumnya, walaupun di-rollback sebelumnya. Snapshot akhir menunjukkan nilai akhir dari sumber daya X (X = 33), dan status transaksi ditampilkan sebagai selesai atau gagal sesuai dengan hasil eksekusi. Program ini memberikan gambaran kompleksitas pengelolaan antrian tunggu dan rollback dalam situasi yang melibatkan lebih dari satu transaksi.

b. Optimistic Concurrency Control (OCC)

Nama Kelas	**Deskripsi**
OCCWriteInstruction	Kelas instruksi yang berisi eksekusi instruksi *write* untuk algoritma OCC
OCCReadInstruction	Kelas instruksi yang berisi eksekusi instruksi *read* untuk algoritma OCC
OCCCommitInstruction	Kelas instruksi yang berisi eksekusi instruksi *commit* untuk algoritma OCC
OCCResourceHandler	Kelas yang mengatur akses *snapshot* setiap *resource* yang ada untuk setiap transaksi; menangani validasi sebelum *write phase*; serta melakukan proses *write phase*
OCCTransactionManager	Kelas yang mengatur eksekusi instruksi oleh berbagai transaksi yang ada
OCCTransaction	Kelas yang menyimpan informasi terkait transaksi pernah yang diterima instruksinya oleh OCCTransactionManager
OCCInstructionReader	Kelas yang membaca *file* masukan dan memberikan objek instruksi (berdasarkan setiap baris yang dibaca pada *file* tersebut) untuk dieksekusi oleh OCCTransactionManager

Berikut merupakan simulasi dari Optimistic Concurrency Control (OCC).

Input	Output

```
# OCC Case:
# Basic OCC test, without any conflict

R T1 X
W T1 X=1
R T2 Y
W T2 Y=2
C T1
C T2
```

```
Input file name: OCC/0

Algorithm:
[0] CANCEL
[1] Two Phase Locking
[2] Optimistic Concurrency Control
[3] Multiversion Timestamp Ordering Concurrency Control

Choice: 2

[INSTRUCTION] Transaction T1 read snapshot of resource X with value 0
[INSTRUCTION] Transaction T1 wrote snapshot of resource X from 0 to 1
[INSTRUCTION] Transaction T2 read snapshot of resource Y with value 0
[INSTRUCTION] Transaction T2 wrote snapshot of resource Y from 0 to 2
[RESOURCE MANAGER] [ Validating transaction T1 before commit ]
[INSTRUCTION] Transaction T1 committed
[RESOURCE MANAGER] [ Validating transaction T2 before commit ]
[INSTRUCTION] Transaction T2 committed
============================================================
[TRANSACTION MANAGER] [ No more instruction received ]
[TRANSACTION MANAGER] Transaction manager stopped with status:
[TRANSACTION MANAGER] { Transaction ID: T1, Start-Timestamp: 1701431917.716371, Validation-Timestamp: 1
701431917.7169383, Finish-Timestamp: 1701431917.7170463, Status: finished }
[TRANSACTION MANAGER] { Transaction ID: T2, Start-Timestamp: 1701431917.7166579, Validation-Timestamp:
1701431917.7171776, Finish-Timestamp: 1701431917.717278, Status: finished }
[RESOURCE MANAGER] [ Resource snapshot ]
[RESOURCE MANAGER] Resource X = 1
[RESOURCE MANAGER] Resource Y = 2
============================================================
```

Penjelasan:

Pada simulasi ini, terdapat dua transaksi, T1 dan T2, yang beroperasi secara bersamaan tanpa adanya konflik. Kedua transaksi tersebut membaca dan menulis pada sumber daya yang berbeda, yaitu X dan Y. Tidak ada kasus dimana transaksi satu mencoba membaca atau menulis sumber daya yang sedang diakses oleh transaksi lainnya.

Hasil output menunjukkan bahwa kedua transaksi berhasil dieksekusi dan dikonfirmasi (committed) tanpa konflik. Setiap transaksi membaca snapshot dari sumber daya yang bersesuaian, menulis perubahan nilai pada snapshot, dan kemudian berhasil dikonfirmasi setelah melalui tahap validasi. Snapshot terakhir dari sumber daya X adalah 1, dan snapshot terakhir dari sumber daya Y adalah 2.

```
# OCC Case:
# Read-Write Conflict

R T1 X
R T2 X
W T1 X=1
C T1
W T2 X=2
C T2
```

```
Input file name: OCC/1

Algorithm:
[0] CANCEL
[1] Two Phase Locking
[2] Optimistic Concurrency Control
[3] Multiversion Timestamp Ordering Concurrency Control

Choice: 2

[INSTRUCTION] Transaction T1 read snapshot of resource X with value 0
[INSTRUCTION] Transaction T2 read snapshot of resource X with value 0
[INSTRUCTION] Transaction T1 wrote snapshot of resource X from 0 to 1
[RESOURCE MANAGER] [ Validating transaction T1 before commit ]
[INSTRUCTION] Transaction T1 committed
[INSTRUCTION] Transaction T2 wrote snapshot of resource X from 0 to 2
[RESOURCE MANAGER] [ Validating transaction T2 before commit ]
[RESOURCE MANAGER] Conflict detected for resource X between transactions T2 and T1
[TRANSACTION MANAGER] [ OCC Validation failed, starting rollback ]
[TRANSACTION MANAGER] Transaction T2 is aborting
[TRANSACTION MANAGER] Trying to rollback transaction T2
[INSTRUCTION] Transaction T2 read snapshot of resource X with value 1
[INSTRUCTION] Transaction T2 wrote snapshot of resource X from 1 to 2
[RESOURCE MANAGER] [ Validating transaction T2 before commit ]
[INSTRUCTION] Transaction T2 committed
============================================================
[TRANSACTION MANAGER] [ No more instruction received ]
[TRANSACTION MANAGER] Transaction manager stopped with status:
[TRANSACTION MANAGER] { Transaction ID: T1, Start-Timestamp: 1701431942.0747714, Validation-Timestamp:
1701431942.075011, Finish-Timestamp: 1701431942.0755608, Status: finished }
[TRANSACTION MANAGER] { Transaction ID: T2, Start-Timestamp: 1701431942.0758197, Validation-Timestamp:
1701431942.0760052, Finish-Timestamp: 1701431942.0760698, Status: finished }
[RESOURCE MANAGER] [ Resource snapshot ]
[RESOURCE MANAGER] Resource X = 2
============================================================
```

Penjelasan:

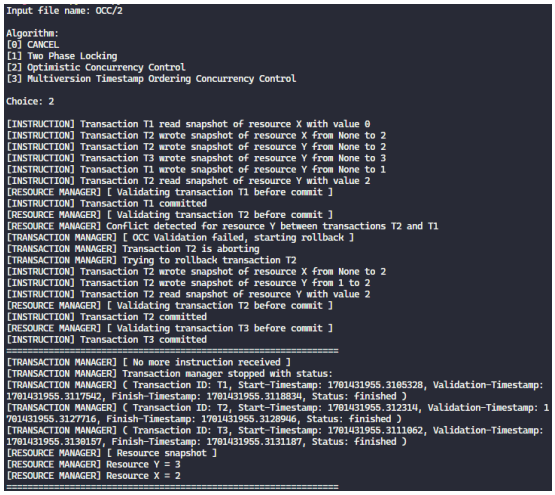
Pada simulasi ini, terjadi konflik antara transaksi T1 dan T2 terkait sumber daya X. Keduanya membaca snapshot dari X pada awalnya, kemudian T1 menulis nilai 1 ke X dan T2 menulis nilai 2 ke X. Sebagai hasilnya, terdapat konflik read-write pada

sumber daya X.

Output menunjukkan bahwa kedua transaksi berhasil dieksekusi dan dikonfirmasi (committed), meskipun ada konflik yang terjadi. Pada OCC, konflik dibalikkan dengan melakukan validasi sebelum commit. Dalam kasus ini, T1 membaca snapshot X sebelum T2 menulis nilainya. Setelah itu, T1 menulis nilainya dan dikonfirmasi, diikuti oleh T2 yang juga dikonfirmasi setelah melalui tahap validasi.

Hasil akhirnya adalah nilai akhir dari sumber daya X adalah 2, yang merupakan nilai yang dihasilkan oleh T2.

```
# OCC Case:
# Intersect transaction

R T1 X
W T2 X=2
W T2 Y=2
W T3 Y=3
W T1 Y=1
R T2 Y
C T1
C T2
C T3
```



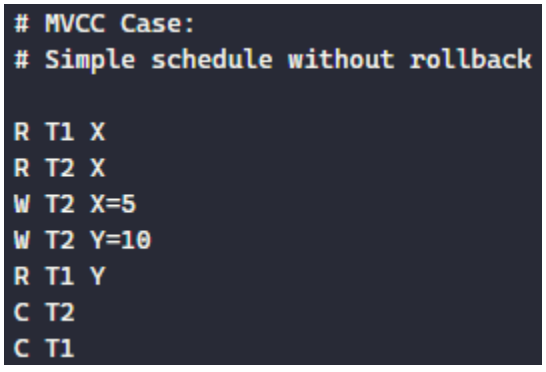
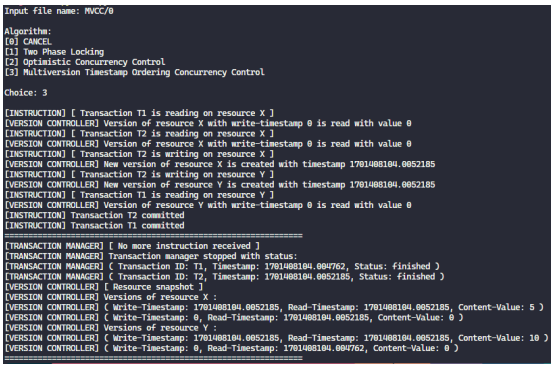
Penjelasan:

Dalam simulasi ini, terjadi interseksi transaksi antara T1, T2, dan T3 pada sumber daya Y. T1 membaca dan menulis nilai 1 ke Y, sementara T2 menulis 2 ke X dan 2 ke Y, dan T3 menulis 3 ke Y. Setelah T1 membaca snapshot Y dari T2, terjadi konflik saat validasi, dan T2 di-rollback dengan dilakukannya retry pada operasi write Y. Proses validasi kembali berhasil, dan T2 committed. T1 dan T3 juga berhasil melakukan commit tanpa konflik. Sebagai hasilnya, nilai Y adalah 3 (dari operasi write T3), dan nilai X adalah 2 (dari operasi write T2). Mekanisme OCC efektif mengelola konflik dengan melakukan rollback dan retry, memastikan konsistensi dalam sistem basis data.

c. Multiversion Timestamp Ordering Concurrency Control (MVCC)

Nama Kelas	Deskripsi
MVCCWriteInstruction	Kelas instruksi yang berisi eksekusi instruksi *write* untuk algoritma MVCC (khususnya MVTO)
MVCCReadInstruction	Kelas instruksi yang berisi eksekusi instruksi *read* untuk algoritma MVCC (khususnya MVTO)
MVCCCommitInstruction	Kelas instruksi yang berisi eksekusi instruksi *commit* untuk algoritma MVCC (khususnya MVTO)
VersionController	Kelas yang mengatur akses dan pembuatan versi *resource*
MVCCTransactionManager	Kelas yang mengatur eksekusi instruksi oleh berbagai transaksi yang ada
MVCCTransaction	Kelas yang menyimpan informasi terkait transaksi pernah yang diterima instruksinya oleh MVCCTransactionManager
MVCCInstructionReader	Kelas yang membaca *file* masukan dan memberikan objek instruksi (berdasarkan setiap baris yang dibaca pada *file* tersebut) untuk dieksekusi oleh MVCCTransactionManager

Berikut merupakan simulasi dari Multiversion Timestamp Ordering Concurrency Control.

Input	Output
	

Penjelasan:

Simulasi ini menggambarkan implementasi MVCC (Multi-Version Concurrency Control) dalam sebuah sistem basis data. Transaksi T1 membaca nilai X dan Y, sementara transaksi T2 membaca nilai X, menulis 5 ke X, dan menulis 10 ke Y. Setiap operasi baca dan tulis menghasilkan versi baru dari sumber daya yang berkaitan

dengan timestamp. Setelah commit, transaksi T1 dan T2 menunjukkan status selesai, dan snapshot akhir dari versi sumber daya X dan Y menunjukkan semua versi yang telah dibuat bersama dengan informasi timestamp dan nilai.

```
# MVCC Case:
# Simple schedule with rollback

R T1 X
R T2 X
W T1 X=5
W T1 X=10
C T2
C T1
```

```
Input file name: MVCC/1

Algorithm:
[0] CANCEL
[1] Two Phase Locking
[2] Optimistic Concurrency Control
[3] Multiversion Timestamp Ordering Concurrency Control

Choice: 3

[INSTRUCTION] [ Transaction T1 is reading on resource X ]
[VERSION CONTROLLER] Version of resource X with write-timestamp 0 is read with value 0
[INSTRUCTION] [ Transaction T2 is reading on resource X ]
[VERSION CONTROLLER] Version of resource X with write-timestamp 0 is read with value 0
[INSTRUCTION] [ Transaction T1 is writing on resource X ]
[TRANSACTION MANAGER] [ Failed executing W(X=5) from transaction T1, starting cascading rollback ]
[TRANSACTION MANAGER] Transaction T1 is aborting
[TRANSACTION MANAGER] Trying to rollback transaction T1
[INSTRUCTION] [ Transaction T1 is reading on resource X ]
[VERSION CONTROLLER] Version of resource X with write-timestamp 0 is read with value 0
[INSTRUCTION] [ Transaction T1 is writing on resource X ]
[VERSION CONTROLLER] New version of resource X is created with timestamp 1701408149.837287
[INSTRUCTION] [ Transaction T1 is writing on resource X ]
[VERSION CONTROLLER] Version of resource X with write-timestamp 1701408149.837287 is updated from 5 to 10
[INSTRUCTION] Transaction T2 committed
[INSTRUCTION] Transaction T1 committed
============================================================
[TRANSACTION MANAGER] [ No more instruction received ]
[TRANSACTION MANAGER] Transaction manager stopped with status:
[TRANSACTION MANAGER] ( Transaction ID: T1, Timestamp: 1701408149.837287, Status: finished )
[TRANSACTION MANAGER] ( Transaction ID: T2, Timestamp: 1701408149.8369734, Status: finished )
[VERSION CONTROLLER] [ Resource snapshot ]
[VERSION CONTROLLER] Versions of resource X :
[VERSION CONTROLLER] ( Write-Timestamp: 1701408149.837287, Read-Timestamp: 1701408149.837287, Content-Value: 10 )
[VERSION CONTROLLER] ( Write-Timestamp: 0, Read-Timestamp: 1701408149.837287, Content-Value: 0 )
============================================================
```

Penjelasan:

Simulasi ini menunjukkan implementasi MVCC dalam sebuah sistem basis data dengan sebuah skenario rollback. Transaksi T1 membaca nilai X, dan kemudian mencoba menulis nilai 5 dan 10 ke X. Namun, operasi penulisan pertama di-rollback karena kegagalan, dan proses cascading rollback dimulai. Setelah di-rollback, T1 berhasil menyelesaikan operasi penulisan ulang ke X dengan nilai 10. Sementara itu, transaksi T2 membaca nilai X dan menyelesaikan operasi commit tanpa masalah. Snapshot akhir menunjukkan versi terakhir dari sumber daya X bersama dengan informasi timestamp dan nilai.

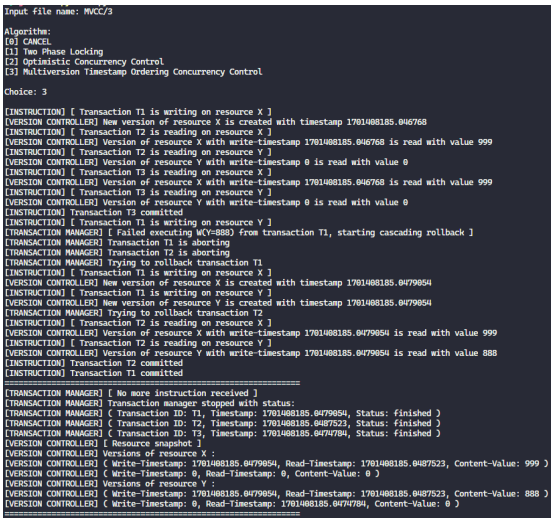
```
# MVCC Case:
# Schedule with cascading rollback

W T1 X=999
R T2 X
R T2 Y
R T3 X
R T3 Y
W T1 Y=888
C T2
C T1
C T3
```

```
Input file name: MVCC/2

Algorithm:
[0] CANCEL
[1] Two Phase Locking
[2] Optimistic Concurrency Control
[3] Multiversion Timestamp Ordering Concurrency Control

Choice: 3

[INSTRUCTION] [ Transaction T1 is writing on resource X ]
[VERSION CONTROLLER] New version of resource X is created with timestamp 1701408166.211253
[INSTRUCTION] [ Transaction T2 is reading on resource X ]
[VERSION CONTROLLER] Version of resource X with write-timestamp 1701408166.211253 is read with value 999
[INSTRUCTION] [ Transaction T2 is reading on resource Y ]
[VERSION CONTROLLER] Version of resource Y with write-timestamp 0 is read with value 0
[INSTRUCTION] [ Transaction T3 is reading on resource X ]
[VERSION CONTROLLER] Version of resource X with write-timestamp 1701408166.211253 is read with value 999
[INSTRUCTION] [ Transaction T3 is reading on resource Y ]
[VERSION CONTROLLER] Version of resource Y with write-timestamp 0 is read with value 0
[INSTRUCTION] [ Transaction T1 is writing on resource Y ]
[TRANSACTION MANAGER] [ Failed executing W(Y=888) from transaction T1, starting cascading rollback ]
[TRANSACTION MANAGER] Transaction T1 is aborting
[TRANSACTION MANAGER] Transaction T3 is aborting
[TRANSACTION MANAGER] Transaction T2 is aborting
[TRANSACTION MANAGER] Trying to rollback transaction T1
[INSTRUCTION] [ Transaction T1 is writing on resource X ]
[VERSION CONTROLLER] New version of resource X is created with timestamp 1701408166.2120185
[INSTRUCTION] [ Transaction T1 is writing on resource Y ]
[VERSION CONTROLLER] New version of resource Y is created with timestamp 1701408166.2120185
[TRANSACTION MANAGER] Trying to rollback transaction T3
[INSTRUCTION] [ Transaction T3 is reading on resource X ]
[VERSION CONTROLLER] Version of resource X with write-timestamp 1701408166.2120185 is read with value 999
[INSTRUCTION] [ Transaction T3 is reading on resource Y ]
[VERSION CONTROLLER] Version of resource Y with write-timestamp 1701408166.2120185 is read with value 888
[TRANSACTION MANAGER] Trying to rollback transaction T2
[INSTRUCTION] [ Transaction T2 is reading on resource X ]
[VERSION CONTROLLER] Version of resource X with write-timestamp 1701408166.2120185 is read with value 999
[INSTRUCTION] [ Transaction T2 is reading on resource Y ]
[VERSION CONTROLLER] Version of resource Y with write-timestamp 1701408166.2120185 is read with value 888
[INSTRUCTION] Transaction T2 committed
[INSTRUCTION] Transaction T1 committed
[INSTRUCTION] Transaction T3 committed
============================================================
[TRANSACTION MANAGER] [ No more instruction received ]
[TRANSACTION MANAGER] Transaction manager stopped with status:
[TRANSACTION MANAGER] ( Transaction ID: T1, Timestamp: 1701408166.2120185, Status: finished )
[TRANSACTION MANAGER] ( Transaction ID: T2, Timestamp: 1701408166.2128925, Status: finished )
[TRANSACTION MANAGER] ( Transaction ID: T3, Timestamp: 1701408166.2127452, Status: finished )
[VERSION CONTROLLER] [ Resource snapshot ]
[VERSION CONTROLLER] Versions of resource X :
[VERSION CONTROLLER] ( Write-Timestamp: 1701408166.2120185, Read-Timestamp: 1701408166.2128925, Content-Value: 999 )
[VERSION CONTROLLER] ( Write-Timestamp: 0, Read-Timestamp: 0, Content-Value: 0 )
[VERSION CONTROLLER] Versions of resource Y :
[VERSION CONTROLLER] ( Write-Timestamp: 1701408166.2120185, Read-Timestamp: 1701408166.2128925, Content-Value: 888 )
[VERSION CONTROLLER] ( Write-Timestamp: 0, Read-Timestamp: 1701408166.2116315, Content-Value: 0 )
============================================================
```

Penjelasan:

Simulasi ini menunjukkan implementasi MVCC dalam sebuah sistem basis data dengan skenario rollback yang melibatkan transaksi T1, T2, dan T3. Transaksi T1 mencoba menulis nilai 999 ke X dan 888 ke Y, tetapi operasi penulisan ke Y gagal, memulai proses cascading rollback yang melibatkan transaksi T2 dan T3. Setelah di-rollback, T1 berhasil menyelesaikan operasi penulisan ulang ke X dan Y dengan nilai yang benar. Sementara itu, T2 dan T3 juga berhasil menyelesaikan operasi baca dan commit mereka. Snapshot akhir menunjukkan versi terakhir dari sumber daya X dan Y bersama dengan informasi timestamp dan nilai.

```
# MVCC Case:
# Unrecoverable schedule

W T1 X=999
R T2 X
R T2 Y
R T3 X
R T3 Y
C T3
W T1 Y=888
C T2
C T1
```



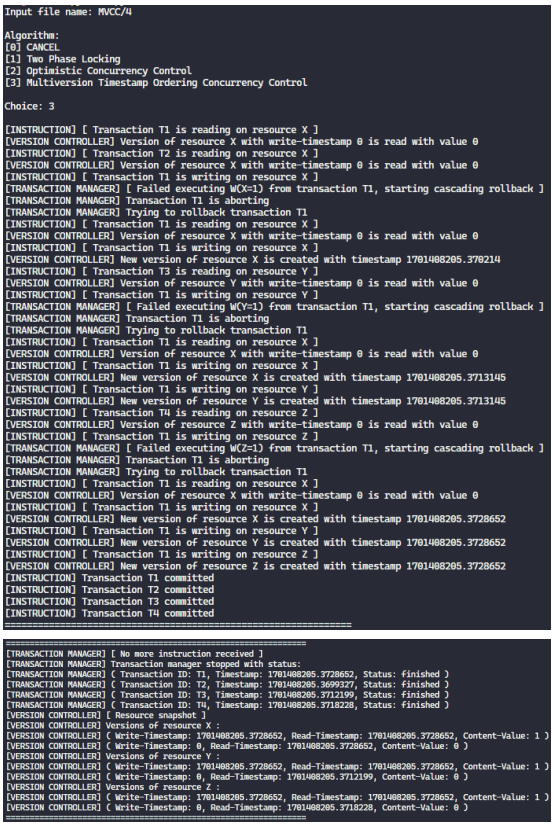
Penjelasan:

Simulasi ini menunjukkan implementasi MVCC dalam sebuah sistem basis data dengan skenario tidak dapat dipulihkan (unrecoverable). Transaksi T1 mencoba menulis nilai 999 ke X dan 888 ke Y, tetapi operasi penulisan ke Y gagal, memulai proses cascading rollback yang melibatkan transaksi T2 dan T3. Namun, transaksi T3 berhasil menyelesaikan operasi commitnya sebelum rollback dimulai, membuatnya menjadi tidak dapat dipulihkan.

Setelah proses tersebut, T1 dan T2 di-rollback, tetapi T3 tetap berhasil menyelesaikan operasi commitnya. Snapshot akhir menunjukkan versi terakhir dari sumber daya X dan Y bersama dengan informasi timestamp dan nilai. Unrecoverable schedule seperti ini terjadi ketika sebuah transaksi sudah selesai dan commit sebelum transaksi lain memulai rollback, sehingga tidak memungkinkan untuk mengembalikan perubahan

yang dilakukan oleh transaksi yang di-rollback.

```
# MVCC Case:
# Multiple rollback on the same instruction

R T1 X
R T2 X
W T1 X=1
R T3 Y
W T1 Y=1
R T4 Z
W T1 Z=1
C T1
C T2
C T3
C T4
```



Penjelasan:

Simulasi ini menunjukkan implementasi MVCC dalam sebuah sistem basis data dengan skenario yang melibatkan rollback yang berulang kali pada instruksi yang sama. Transaksi T1 awalnya mencoba menulis nilai 1 ke X, Y, dan Z, tetapi operasi penulisan Y dan Z gagal, memulai proses cascading rollback. Proses rollback terjadi sebanyak dua kali untuk setiap operasi penulisan yang gagal.

Setelah proses tersebut, T1, T2, T3, dan T4 berhasil menyelesaikan operasi commit masing-masing. Snapshot akhir menunjukkan versi terakhir dari sumber daya X, Y, dan Z bersama dengan informasi timestamp dan nilai. Ini menunjukkan bahwa, meskipun ada beberapa rollback, transaksi berhasil diselesaikan dan membuat versi terakhir dari sumber daya yang sesuai.

3. Eksplorasi Recovery

a. *Write-Ahead Log*

Metode *recovery* Write-Ahead Log (WAL) pada postgres berfungsi dengan cara mencatat semua perubahan yang terjadi pada *database* sebelum perubahan diterapkan pada data asli di *disk*. Proses ini dimulai saat terdapat permintaan perubahan pada *database*. Postgres akan mencatat semua detail perubahan tersebut dalam sebuah *log* transaksi yang disebut Write-Ahead Log (WAL) sebelum perubahan benar-benar dilakukan. Secara *default*, *log* ini disimpan dalam direktori data/pg_wal di dalam direktori data dari instalasi postgres.

Dengan adanya Write-Ahead Log (WAL) postgres dapat menggunakan informasi yang tercatat pada *log* untuk melakukan pemulihan data. Hal ini dilakukan dengan menerapkan kembali perubahan-perubahan yang tercatat dalam *log* sehingga *database* dapat kembali ke keadaan sebelum terjadinya *crash*.

b. *Continuous Archiving*

Continuous Archiving adalah proses pengarsipan file *log* Write-Ahead Log (WAL) ke penyimpanan eksternal yang lebih stabil secara teratur. Hal ini memastikan bahwa *log* tersebut tidak hanya tersimpan di lokasi internal server, namun juga di lokasi eksternal yang aman. Jika terjadi kegagalan pada sistem, file *log* yang diarsipkan ini dapat digunakan untuk pemulihan data yang lebih menyeluruh.

c. *Point-in-Time Recovery*

Point In Time recovery (PITR) adalah teknik yang memungkinkan melakukan *recovery database* dengan mengembalikan *database* ke keadaan tertentu pada titik waktu spesifik di masa lampau. Ketika menggunakan PITR, informasi yang tercatat dalam *log* WAL yang telah dicadangkan secara berkala melalui Continuous Archiving dapat digunakan untuk mengembalikan *database* ke keadaan yang sesuai dengan titik waktu tertentu di masa lampau. Dengan cara ini, PITR memberikan fleksibilitas untuk melakukan pemulihan data dengan presisi pada titik waktu yang diinginkan dalam *history* perubahan *database*.

d. Simulasi Kegagalan pada PostgreSQL

Berikut ini adalah langkah-langkah yang digunakan dalam mensimulasikan kegagalan pada postgres serta pemulihan data yang dilakukan pada sistem operasi windows.

1. Metode recovery yang digunakan adalah Point In Time Recovery. Sebelum menggunakan metode ini kita perlu mengaktifkan Continuous Archiving.

2. Buat folder untuk menyimpan pengarsipan dari file Write Ahead Log (WAL). Misal dengan nama direktori ‘walbackup’.

3. Pengaktifan proses Continuous Archiving dilakukan dengan mengubah parameter pada file konfigurasi ‘postgresql.conf’. File ini terletak pada direktori instalasi postgres pada folder ‘data’. Parameter dapat diubah dengan melakukan pengeditan langsung pada file ‘postgresql.conf’ atau melalui *query* ‘ALTER SYSTEM’ pada postgres. Adapun parameter yang perlu diubah adalah sebagai berikut.

   ```
   ALTER SYSTEM SET wal_level = replica;
   ```

   Pengaturan ‘wal_level’ menjadi nilai ‘replica’ atau level yang lebih tinggi memungkinkan postgres untuk mencatat informasi transaksi yang diperlukan untuk tujuan replikasi.

   ```
   ALTER SYSTEM archive_mode = on;
   ```

   Saat 'archive_mode' diatur ke 'on', fitur Continuous Archiving diaktifkan, memungkinkan sistem untuk secara terus-menerus menyimpan log transaksi.

   ```
   ALTER SYSTEM archive_command = ‘copy “%p” “path\\walbackup\\%f”’
   ```

   Parameter 'archive_command' digunakan untuk menentukan perintah yang akan menyalin file *log* transaksi (WAL). Dengan menggunakan nilai ini, postgres akan secara otomatis menyalin file *log* WAL ke direktori ‘walbackup’ untuk penyimpanan arsip transaksi yang diperlukan untuk pemulihan

   ```
   ALTER SYSTEM restore_command = ‘copy “path\\walbackup\\%f” “%p”’
   ```

   Selain itu, terdapat pengubahan parameter ‘restore_command’ yang akan menentukan perintah untuk memulihkan file WAL pada proses pemulihan. Dalam

kasus Point In Time Recovery (PITR), perintah ini mendefinisikan bagaimana sistem akan mendapatkan file WAL yang diperlukan untuk mengembalikan data ke titik waktu tertentu.

4. Pastikan bahwa perubahan parameter konfigurasi telah diaplikasikan dengan menjalankan *query* 'SHOW' pada postgres. Jika tidak terdapat perubahan, lakukan *restart* pada *service* postgres.

```
postgres=# show wal_level;
 wal_level
-----------
 replica
(1 row)
```

```
postgres=# show archive_mode;
 archive_mode
--------------
 on
(1 row)
```

```
postgres=# show archive_command;
          archive_command
-------------------------------------
 copy "%p" "C:\\tmp\\walbackup\\%f"
(1 row)
```

```
postgres=# show restore_command;
          restore_command
-------------------------------------
 copy "C:\\tmp\\walbackup\\%f" "%p"
(1 row)
```

5. Simulasi kegagalan akan dilakukan pada relasi 'department' di dalam basis data 'Hospital' dengan kondisi awal sebagai berikut.

```
hospital=# \dt
              List of relations
 Schema |      Name       | Type  |  Owner
--------+-----------------+-------+----------
 public | affiliated_with | table | postgres
 public | appointment     | table | postgres
 public | block           | table | postgres
 public | department      | table | postgres
 public | medication      | table | postgres
 public | nurse           | table | postgres
 public | on_call         | table | postgres
 public | patient         | table | postgres
 public | physician       | table | postgres
 public | prescribes      | table | postgres
 public | procedures      | table | postgres
 public | room            | table | postgres
 public | stay            | table | postgres
 public | trained_in      | table | postgres
 public | undergoes       | table | postgres
```

```
hospital=# select * from department;
 departmentid |          name           | head
--------------+-------------------------+------
            1 | General Medicine        |    4
            2 | Surgery                 |    7
            3 | Psychiatry              |    9
            4 | Accident and Emergency  |   15
            5 | Anaesthetics            |   20
            6 | Cardiology              |   65
            7 | Diagnostic Imaging      |   37
            8 | Ear Nose and Throat     |   95
            9 | Gastroenterology        |   88
(9 rows)
```

6. Buat direktori baru yang akan digunakan untuk menyimpan salinan data postgres menggunakan perintah berikut pada *command prompt.*

   `pg_basebackup -U username -D "path\databackup"`

   Perintah ini bertujuan untuk membuat salinan dari data postgres yang akan disimpan pada direktori 'databackup'.

7. Lakukan perubahan pada *database*, misalnya penambahan data pada relasi 'department'.

```
hospital=# insert into department (departmentid,name,head) values (10, 'Neurology', 1);
INSERT 0 1
```

```
hospital=# select * from department;
 departmentid |          name          | head
--------------+------------------------+------
            1 | General Medicine       |    4
            2 | Surgery                |    7
            3 | Psychiatry             |    9
            4 | Accident and Emergency |   15
            5 | Anaesthetics           |   20
            6 | Cardiology             |   65
            7 | Diagnostic Imaging     |   37
            8 | Ear Nose and Throat    |   95
            9 | Gastroenterology       |   88
           10 | Neurology              |    1
(10 rows)
```

8. Simulasikan terjadinya kegagalan sistem dengan menghapus semua file yang ada pada direktori ‘data’ yang terletak pada direktori instalasi postgres. Setelah itu, coba untuk *restart* pada *service* postgres (seharusnya *service* tidak dapat dijalankan).

9. Salin data *backup* pada direktori ‘databackup’, yang telah dibuat pada langkah enam. Kemudian jalankan kembali *service* postgres dan buka kembali postgres. Berikut ini adalah tabel ‘department’ setelah terjadinya crash, dimana perubahan penambahan data department tidak terekam.

```
hospital=# select * from department;
 departmentid |          name          | head
--------------+------------------------+------
            1 | General Medicine       |    4
            2 | Surgery                |    7
            3 | Psychiatry             |    9
            4 | Accident and Emergency |   15
            5 | Anaesthetics           |   20
            6 | Cardiology             |   65
            7 | Diagnostic Imaging     |   37
            8 | Ear Nose and Throat    |   95
            9 | Gastroenterology       |   88
(9 rows)
```

Catatan: Saat melihat kondisi database pada saat terjadi crash, file Write-Ahead Log (WAL) akan tertimpa oleh *log* yang baru. Ini dapat mengakibatkan kegagalan dalam proses pemulihan. Oleh karena itu, disarankan untuk mencadangkan file WAL yang telah diarsipkan sebelum memeriksa kondisi *database*. Pastikan untuk

mengembalikan direktori 'walbackup' dan direktori ‘data’ ke kondisi yang sama seperti setelah terjadinya *crash*.

10. Untuk mengembalikan data yang hilang, langkah awalnya adalah membuat file ‘recovery.signal’ yang akan memberikan sinyal pada postgres untuk melakukan proses pemulihan (*recovery*). Proses *recovery* ini akan menjalankan perintah dari ‘restore_command’ yaitu dengan menyalin arsip Write-Ahead Log (WAL) yang ada dalam direktori 'walbackup' ke dalam direktori 'data/pg_wal'. Dengan melakukan langkah-langkah ini, postgres akan menjalankan perintah-perintah yang sebelumnya tidak tercatat, sehingga basis data dapat dikembalikan ke keadaan sebelum terjadinya kegagalan. *Restart* kembali *service* postgres untuk menjalankannya kembali. Keberhasilan *recovery* dapat dilihat pada *folder* ‘data/log’ sebagai berikut.

```
LOG:  starting archive recovery
LOG:  restored log file "000000010000000000000030" from archive
LOG:  redo starts at 0/30000060
LOG:  consistent recovery state reached at 0/30000138
LOG:  restored log file "000000010000000000000031" from archive
LOG:  restored log file "000000010000000000000032" from archive
LOG:  restored log file "000000010000000000000033" from archive
LOG:  redo done at 0/33000060 system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.19 s
LOG:  last completed transaction was at log time 2023-12-01 01:35:42.73273+07
LOG:  restored log file "000000010000000000000033" from archive
LOG:  selected new timeline ID: 2
LOG:  archive recovery complete
```

Berikut adalah tabel ‘department’ yang berhasil dipulihkan.

```
hospital=# select * from department;
 departmentid |          name          | head
--------------+------------------------+------
            1 | General Medicine       |    4
            2 | Surgery                |    7
            3 | Psychiatry             |    9
            4 | Accident and Emergency |   15
            5 | Anaesthetics           |   20
            6 | Cardiology             |   65
            7 | Diagnostic Imaging     |   37
            8 | Ear Nose and Throat    |   95
            9 | Gastroenterology       |   88
           10 | Neurology              |    1
(10 rows)
```

Catatan: Jika ingin memulihkan (*restore*) basis data ke waktu tertentu, dapat dilakukan dengan mengubah salah satu parameter 'recovery target'.

4. Pembagian Kerja

NIM	Nama	Bagian
13521058	Ghazi Akmal Fauzan	Implementasi OCC
13521068	Ilham Akbar	Eksplorasi Transaction Isolation
13521130	Althaaf Khasyi Atisomya	Eksplorasi Recovery
13521148	Johanes Lee	Implementasi 2PL, OCC, dan MVCC

Link github: https://github.com/Enliven26/Tubes-IF3140

Referensi

Arora, Dhruv. "Transaction Isolation in Postgres." Medium. Diakses pada 1 Desember 2023, dari https://medium.com/@darora8/transaction-isolation-in-postgres-ec4d34a65462

PostgreSQL. (n.d.). .26.3. Continuous Archiving and Point-in-Time Recovery (PITR). Diakses pada 1 Desember 2023, dari https://www.postgresql.org/docs/current/continuous-archiving.html

PostgreSQL. (n.d.). 13.2. Isolasi Transaksi. Diakses pada 1 Desember 2023, dari https://www.postgresql.org/docs/current/transaction-iso.html

PostgreSQL. (n.d.). .30.3. Write-Ahead Logging (WAL). Diakses pada 1 Desember 2023, dari https://www.postgresql.org/docs/current/wal-intro.html

Tim Pengajar IF3140 Semester 1 2022/2023 Concurrency Control , “Concurrency Control Bagian 1”, https://cdn-edunex.itb.ac.id/54401-Database-Management-Parallel-Class/208268-Minggu-10/1698209953268_IF3140-2023-W10_1---Concurrency-Control-part-1.pdf, diakses pada 29 November 2023

Tim Pengajar IF3140 Semester 1 2022/2023 Concurrency Control , “Concurrency Control Bagian 2”, https://cdn-edunex.itb.ac.id/54401-Database-Management-Parallel-Class/209436-Minggu-11/1698824568964_IF3140-2023-W11_1---Concurrency-Control-part-2.pdf , diakses pada 29 November 2023

Tim Pengajar IF3140 Semester 1 2022/2023 Concurrency Control , “Concurrency Control Bagian 3”, https://cdn-edunex.itb.ac.id/54401-Database-Management-Parallel-Class/210572-Minggu-12/1699497560966_IF3140-2023-W12_1---Concurrency-Control-part-3.pdf, diakses pada 29 November 2023