

**TUGAS BESAR 2**  
**IF3140 MANAJEMEN BASIS DATA**  
**MEKANISME CONCURRENCY CONTROL DAN RECOVERY**



Disusun oleh:

Hasbi Al Farabi	10022124
Muhammad Risqi Firdaus	13520043
Jevant Jedia Augustine	13520133
Willy Wilsen	13520160

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**2022**

# **Daftar Isi**

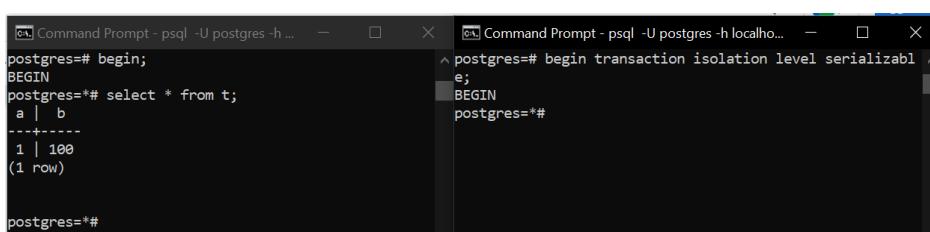
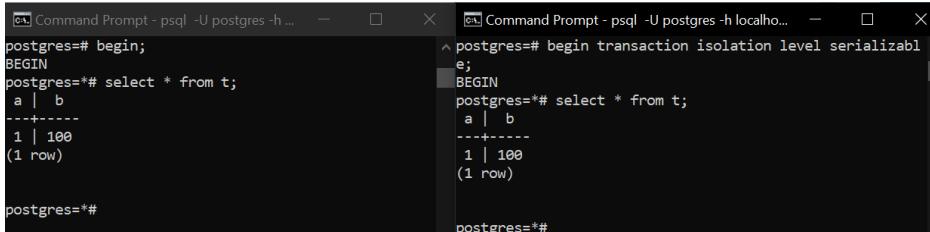
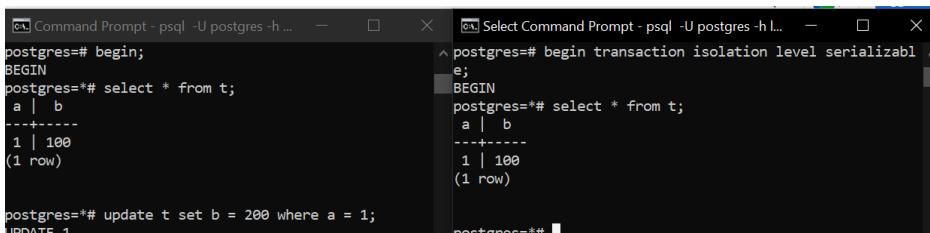
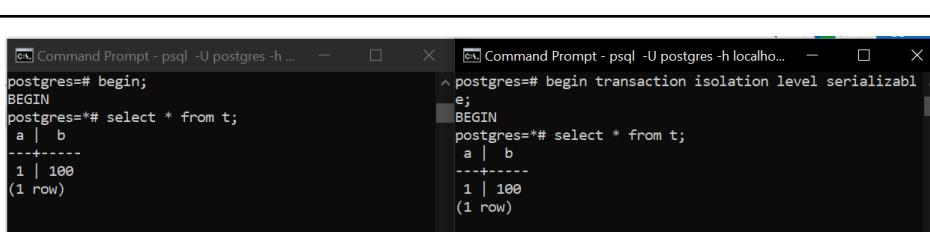
<b>Daftar Isi</b>	<b>2</b>
<b>Eksplorasi Concurrency Control</b>	<b>3</b>
Serializability	3
Repeatable Read	4
Read Committed	5
Read Uncommitted	6
<b>Implementasi Concurrency Control Protocol</b>	<b>7</b>
Simple Locking (Exclusive Locks Only)	7
Serial Optimistic Concurrency Control (OCC)	8
Multiversion Timestamp Ordering Concurrency Control (MVCC)	10
<b>Eksplorasi Recovery</b>	<b>12</b>
Write-Ahead Log	12
Continuous Archiving	13
Point-in-Time Recovery	14
<b>Kesimpulan dan Saran</b>	<b>16</b>
Kesimpulan	16
Saran	16
<b>Pembagian Kerja</b>	<b>17</b>
<b>Referensi</b>	<b>18</b>

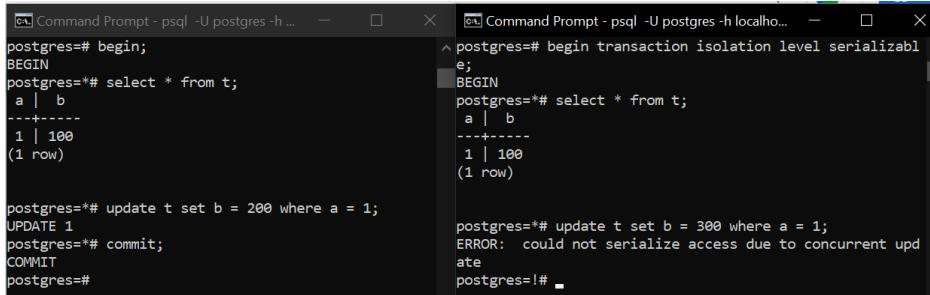
# Eksplorasi Concurrency Control

## Serializability

Memastikan eksekusi yang berlangsung *serializable*, yaitu eksekusi dengan jadwal yang ekuivalen dengan jadwal serial.

Contoh *schedule* dan simulasi:

T1	T2	Simulasi
Read(X)		 A screenshot showing two terminal windows. The left window (T1) contains a 'begin' statement, followed by a 'select * from t' query which returns a single row 'a   b' with value '1   100'. The right window (T2) contains a 'begin transaction isolation level serializable' statement, followed by a 'BEGIN' statement.
	Read(X)	 A screenshot showing the same two terminal windows. Both T1 and T2 have executed their respective 'select * from t' statements, both returning the same row 'a   b' with value '1   100'.
Write(X)		 A screenshot showing the same two terminal windows. T1 has performed an 'update t set b = 200 where a = 1;' operation, resulting in 'UPDATE 1' rows affected. T2 has then read the updated row, returning 'a   b' with value '1   200'.
	Write(X)	 A screenshot showing the same two terminal windows. Both T1 and T2 have performed 'update t set b = 200 where a = 1;' operations, each resulting in 'UPDATE 1' rows affected. T2's update overwrote T1's, so when T2 reads the row, it returns 'a   b' with value '1   300'.

Commit		
--------	--	------------------------------------------------------------------------------------

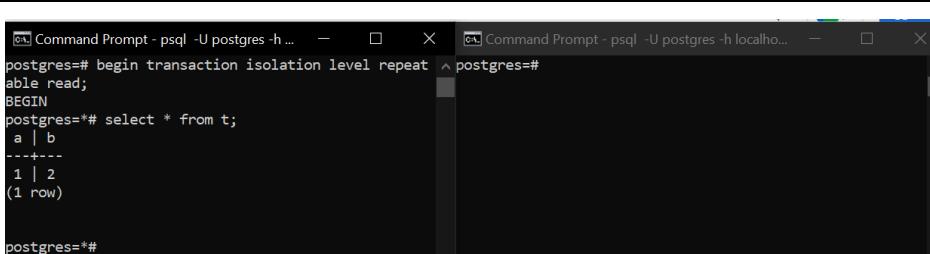
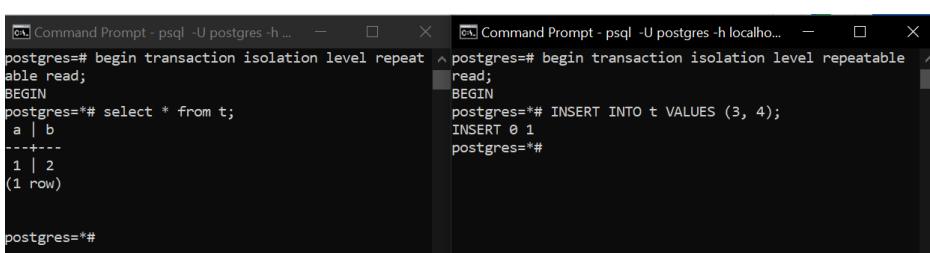
Penjelasan:

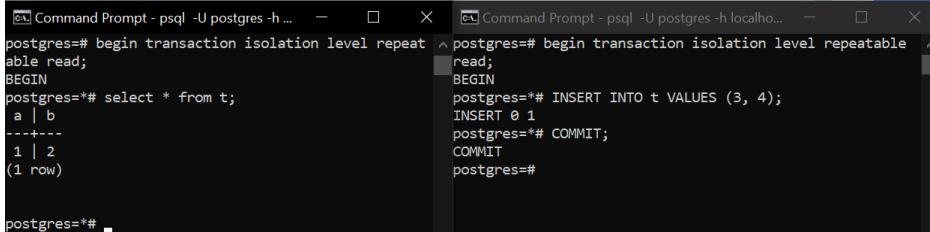
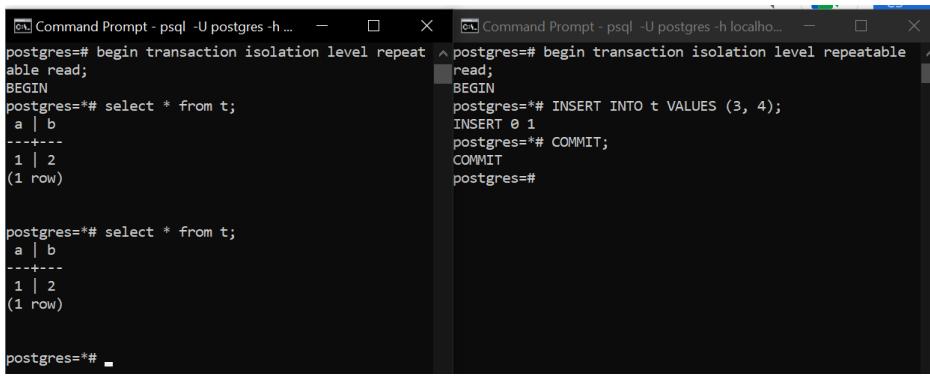
Contoh *schedule* yang diberikan merupakan sebuah contoh jadwal yang tidak serializable. Pada saat jadwal tersebut disimulasikan, dapat dilihat bahwa pada saat T1 commit, terdapat pesan *error* pada T2 yang mengatakan bahwa operasi yang dilakukan tidak bisa di-*serialize* sehingga T2 harus di-*rollback* agar operasi yang dilakukan di T2 dapat dilakukan.

## Repeatable Read

Memungkinkan hanya data yang sudah di-*commit* untuk dibaca. Antara dua pembacaan data oleh transaksi, tidak ada transaksi lain yang diizinkan untuk memperbarui data tersebut. Dapat saja tidak *serializable*.

Contoh *schedule* dan simulasi:

T1	T2	Simulasi
Read(X)		
	Write(X)	

	Commit	
	Read(X)	

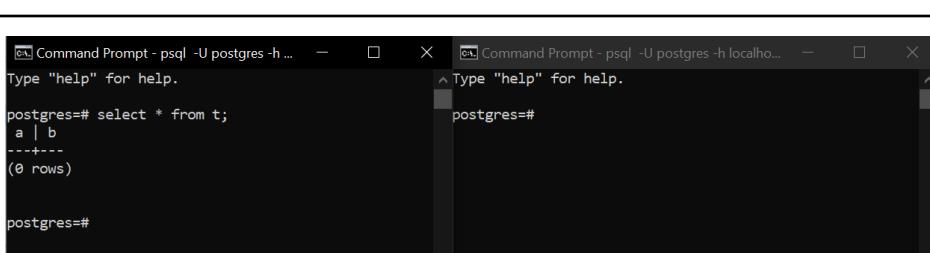
Penjelasan:

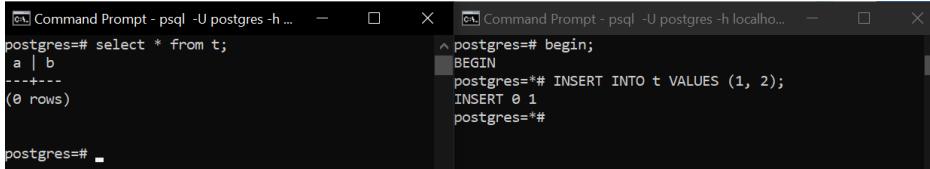
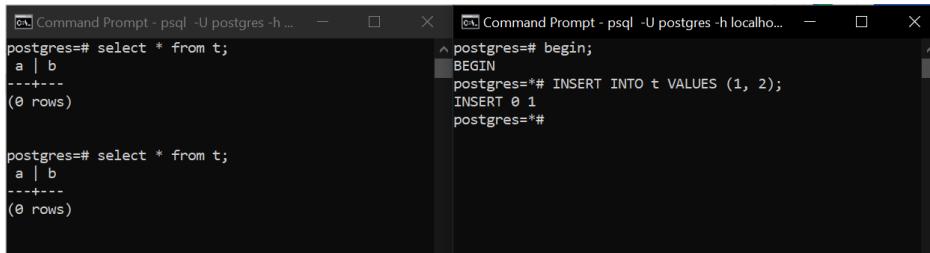
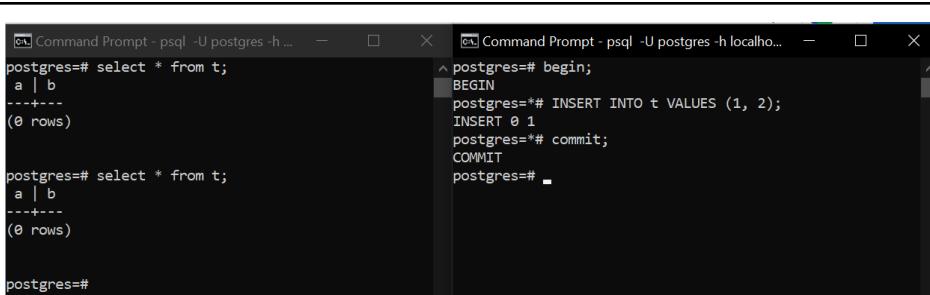
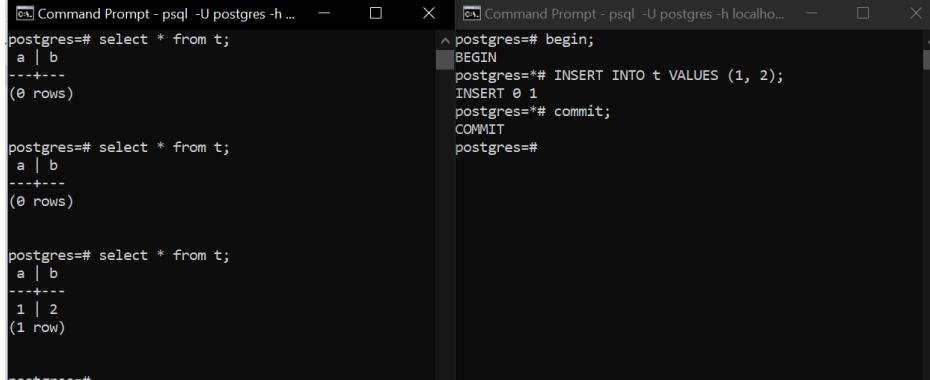
Simulasi menunjukkan bahwa hasil dari Read(X) pada T1 baik sesudah maupun sebelum T2 melakukan operasi Write(X) dan COMMIT adalah sama yang menunjukkan isolasi level *Repeatable Read* dimana antara 2 pembacaan data X oleh dua transaksi, hasil pembacaan data X pasti sama.

## Read Committed

Hanya memperbolehkan data yang sudah di-*commit* untuk dibaca, tetapi tidak membutuhkan *repeatable read*.

Contoh *schedule* dan simulasi:

T1	T2	Simulasi
Read(X)		

	Write(X)	
	Read(X)	
	Commit	
	Read(X)	

Penjelasan:

Pada simulasi dapat dilihat bahwa hasil Read(X) pada T1 hanya akan menampilkan nilai X awal walaupun T2 melakukan Write pada data X. Read(X) pada T1 hanya akan menampilkan nilai X yang telah diubah oleh T2 sesudah T2 melakukan *commit*.

## Read Uncommitted

Memperbolehkan data yang belum di-*commit* untuk dibaca.

# Implementasi Concurrency Control Protocol

## Simple Locking (Exclusive Locks Only)

Simple Locking concurrency control akan memberikan *exclusive lock* kepada transaksi yang ingin mengolah sebuah data item, baik itu melakukan write atau read. Algoritma simple locking akan melakukan iterasi pada jadwal masukan dan mengolah operasi yang diiterasikan. Berdasarkan kondisi pada saat itu, ada beberapa hal yang akan dilakukan, antara lain: memberikan kunci, melakukan commit/melepas kunci, memasukan ke queue, dan mengeluarkan dari queue. Algoritma simple locking dibuat dengan menggunakan class SimpleLock.

Kelas SimpleLock akan menerima masukan berupa nama file yang isinya berupa jadwal masukan. Isi dari file tersebut akan diubah menjadi array yang dapat diolah oleh program. Class SimpleLock memiliki sebuah fungsi utama getNewSchedule yang akan memberikan jadwal dengan lock yang bersesuaian serta beberapa fungsi komplementer yang membantu lajunya fungsi utama tersebut. Hasil akhir dari jadwal yang telah diberi kunci disimpan dalam bentuk array dan dapat dicetak dengan menggunakan fungsi printFinal.

Pengujian:

1	R1(X),R2(Y),R2(X),R1(Y),C1,C2 (deadlock)
<pre>C:\ITB\Informatika\Sem 5\MBD\Tubes 2\Concurrency-Control\simplelock&gt;py simple_locking.py Masukkan file input:gagal.txt Initial Schedule = ['R1(X)', 'R2(Y)', 'R2(X)', 'R1(Y)', 'C1', 'C2'] Final Schedule = ['XL1(X)', 'R1(X)', 'XL2(Y)', 'R2(Y)', 'deadlock']</pre>	
2	R1(X),R2(X),C1,R2(Y),C2 (sukses)
<pre>C:\ITB\Informatika\Sem 5\MBD\Tubes 2\Concurrency-Control\simplelock&gt;py simple_locking.py Masukkan file input:sukses1.txt Initial Schedule = ['R1(X)', 'R2(X)', 'C1', 'R2(Y)', 'C2'] Final Schedule = ['XL1(X)', 'R1(X)', 'C1', 'XL2(X)', 'R2(X)', 'XL2(Y)', 'R2(Y)', 'C2']</pre>	
3	R1(A),W1(A),R2(A),W2(A),R1(B),W1(B),R2(B),W2(B),C1,C2 (sukses)
<pre>C:\ITB\Informatika\Sem 5\MBD\Tubes 2\Concurrency-Control\simplelock&gt;py simple_locking.py Masukkan file input:sukses2.txt Initial Schedule = ['R1(A)', 'W1(A)', 'R2(A)', 'W2(A)', 'R1(B)', 'W1(B)', 'R2(B)', 'W2(B)', 'C1', 'C2'] Final Schedule = ['XL1(A)', 'R1(A)', 'W1(A)', 'XL1(B)', 'R1(B)', 'W1(B)', 'C1', 'XL2(A)', 'R2(A)', 'W2(A)', 'XL2(B)', 'R2(B)', 'W2(B)', 'C2']</pre>	

Dapat dilihat pada gambar bahwa hasil yang diharapkan sudah sesuai dengan yang diharapkan. Lock exclusive yang diberikan telah berada pada tempat yang sesuai dan keadaan deadlock telah dideteksi dengan baik.

## Serial Optimistic Concurrency Control (OCC)

Pada serial optimistic concurrency control, thread eksekusi thread dibagi berdasarkan 3 jenis timestamp, yakni start timestamp, validation, dan end timestamp. Program dibuat menggunakan dua buah class Transaction dan class OCC.

Class Transaction berperan sebagai sebuah thread. Class ini menyimpan daftar read, write, serta daftar operasi pada thread tersebut. Selain itu, class ini juga menyimpan ts\_start, ts\_validation (timestamp), dan ts\_end.

Class OCC berisi proses dari read data dari file hingga validasi tiap proses, jika apakah abort. Proses ini menyimpan list of schedule, dan list of thread yang digunakan. Untuk menjalankan program OCC, dapat dipanggil pada main module method OCC dari object class OCC.

Pengujian:

1.	R0A,R2A,W1B,R2C,C2,W0C,C1,C0 (sukses)
<pre>Committing thread 2 Validation succeeded for thread 2  Thread 0 is writing C  Committing thread 1 Validation succeeded for thread 1  Committing thread 0 Validation succeeded for thread 0  ====Printing Status====  Transaction 0: Start: 1670224982.765094 Validation: 1670224983.266988 End: 1670224983.2670312 Read: ['A'] Write: ['C']  Transaction 2: Start: 1670224982.9657786 Validation: 1670224983.1666002 End: 1670224983.1666987 Read: [] Write: ['B']  Transaction 1: Start: 1670224982.865403 Validation: 1670224983.0661993 End: 1670224983.0662842 Read: ['A', 'C'] Write: []</pre>	
2.	R0A,R2A,W1B,R2C,C2,W0B,C1,C0 (sukses)

```
Thread 2 is reading C
Committing thread 2
Validation succeeded for thread 2

Thread 0 is writing B
Committing thread 1
Validation succeeded for thread 1

Committing thread 0
Validation succeeded for thread 0

====Printing Status====

Transaction 0:
Start: 1670225192.4250715
Validation: 1670225192.927157
End: 1670225192.9272866
Read: ['A']
Write: ['B']

Transaction 2:
Start: 1670225192.625895
Validation: 1670225192.8268638
End: 1670225192.8269632
Read: []
Write: ['B']

Transaction 1:
Start: 1670225192.5254502
Validation: 1670225192.7263901
End: 1670225192.7264798
Read: ['A', 'C']
Write: []
```

### 3. R0B,R2A,W1B,R2C,C2,W0B,C1,C0 (gagal)

```
Committing thread 2
Validation succeeded for thread 2

Thread 0 is writing B
Committing thread 1
Validation succeeded for thread 1

Committing thread 0
Validation failed for thread 0
Aborted variables: ['B']

====Printing Status====

Transaction 0:
Start: 1670225291.847016
Validation: 1670225292.349328
End: None
Read: ['B']
Write: ['B']

Transaction 2:
Start: 1670225292.0479674
Validation: 1670225292.2489252
End: 1670225292.2490125
Read: []
Write: ['B']

Transaction 1:
Start: 1670225291.9474828
Validation: 1670225292.1483772
End: 1670225292.1484597
Read: ['A', 'C']
Write: []
```

4.	R0X,R1Y,R2Z,W0X,C1,C2,C0 (sukses)
	<pre> Commiting thread 1 Validation succeeded for thread 1  Commiting thread 2 Validation succeeded for thread 2  Commiting thread 0 Validation succeeded for thread 0  ====Printing Status==== Transaction 0: Start: 1670225381.23219 Validation: 1670225381.733993 End: 1670225381.734119 Read: ['X'] Write: ['X']  Transaction 1: Start: 1670225381.3326564 Validation: 1670225381.5333304 End: 1670225381.5334172 Read: ['Y'] Write: []  Transaction 2: Start: 1670225381.4329631 Validation: 1670225381.6336071 End: 1670225381.6337094 Read: ['Z'] Write: [] </pre>

Algoritma OCC yang diterapkan dapat menyelesaikan proses dengan algoritma OCC sebagaimana mekanisme yang telah dijelaskan sebelumnya. Algoritma ini mengecek validitas timestamp serta operasi yang dilakukan ketika commit. Dari 4 tes yang dilakukan, seluruh testcase diselesaikan dengan tepat.

## Multiversion Timestamp Ordering Concurrency Control (MVCC)

Multiversion Timestamp Ordering concurrency control adalah metode yang menyediakan akses bersamaan ke database dengan menggunakan stempel waktu (**TS**) dan ID transaksi untuk mencapai konsistensi transaksi. MVCC memastikan transaksi (**T**) tidak perlu menunggu untuk membaca objek database (**P**) dengan mempertahankan beberapa versi objek. Setiap versi objek **P** memiliki Read Timestamp(**RTS**) dan Write Timestamp() yang memungkinkan transaksi tertentu **T<sub>i</sub>** membaca versi terbaru dari objek yang mendahului read timestamp **RTS(T<sub>i</sub>)** transaksi.

Jika transaksi (**T<sub>i</sub>**) ingin menulis ke objek, dan transaksi memiliki Timestamp(**TS**) yang lebih awal dari Read Timestamp objek saat ini, **TS(T<sub>i</sub>) < RTS(P)**, maka transaksi dibatalkan dan dimulai ulang. Jika tidak, **T<sub>i</sub>** membuat versi baru objek **P** dan mengatur waktu baca/tulis **TS** dari versi baru ke stempel waktu transaksi **TS ← TS(T<sub>i</sub>)**.

Kelas MVCC akan menerima masukan berupa nama file yang isinya berupa jadwal masukan. Isi dari file tersebut akan diubah menjadi array bertipe queue yang dapat diolah oleh program. Class MVCC memiliki sebuah fungsi utama read dan write yang akan mengolah read timestamp dan write timestamp dari objek pada transaksi yang terjadi. Hasil akhir dari keseluruhan timestamp dapat dicetak dengan menggunakan fungsi printMVCC.

## Pengujian:

1	R1(A),R2(A),R3(B),R1(B),W3(C),W2(C),R1(C),C1,R2(D),W3(B),C3,W2(D),C2
	<pre>PS C:\Users\lenovo\Documents\Concurrency-Control\mvcc&gt; py mvcc.py Masukkan file input: test1.txt Schedule = ['R1(A)', 'R2(A)', 'R3(B)', 'R1(B)', 'W3(C)', 'W2(C)', 'R1(C)', 'C1', 'R2(D)', 'W3(B)', 'C3', 'W2(D)', 'C2'] R1(A) -&gt;    TS(A0) = (1, 0) R2(A) -&gt;    TS(A0) = (2, 0) R3(B) -&gt;    TS(B0) = (3, 0) R1(B) -&gt;    TS(B0) = (3, 0) W3(C) -&gt;    TS(C3) = (3, 3) W2(C) -&gt;    TS(T2) &lt; R-TS(C3) -&gt; rollback Ulang T2 dengan TS = 4 TS(A0) = (4, 0) TS(C2) = (4, 4) R1(C) -&gt;    TS(C0) = (1, 0) R2(D) -&gt;    TS(D0) = (4, 0) W3(B) -&gt;    TS(B3) = (3, 3) W2(D) -&gt;    TS(D2) = (4, 4)</pre>
2	R5(X),R2(Y),R1(Y),W3(Y),W3(Z),R5(Z),R2(Z),R1(X),R4(W),W3(W),W5(Y),W5(Z)
	<pre>PS C:\Users\lenovo\Documents\Concurrency-Control\mvcc&gt; py mvcc.py Masukkan file input: test2.txt Schedule = ['R5(X)', 'R2(Y)', 'R1(Y)', 'W3(Y)', 'W3(Z)', 'R5(Z)', 'R2(Z)', 'R1(X)', 'R4(W)', 'W3(W)', 'W5(Y)', 'W5(Z)'] R5(X) -&gt;    TS(X0) = (5, 0) R2(Y) -&gt;    TS(Y0) = (2, 0) R1(Y) -&gt;    TS(Y0) = (2, 0) W3(Y) -&gt;    TS(Y3) = (3, 3) W3(Z) -&gt;    TS(Z3) = (3, 3) R5(Z) -&gt;    TS(Z3) = (5, 3) R2(Z) -&gt;    TS(Z0) = (2, 0) R1(X) -&gt;    TS(X0) = (5, 0) R4(W) -&gt;    TS(W0) = (4, 0) W3(W) -&gt;    TS(T3) &lt; R-TS(W0) -&gt; rollback Ulang T3 dengan TS = 6 TS(Y3) = (6, 6) TS(Z3) = (6, 6) TS(W3) = (6, 6) W5(Y) -&gt;    TS(T5) &lt; R-TS(Y3) -&gt; rollback Ulang T5 dengan TS = 7 TS(X0) = (7, 0) TS(Z3) = (7, 6) TS(Y5) = (7, 7) W5(Z) -&gt;    TS(Z5) = (7, 7)</pre>

Algoritma MVCC yang diterapkan dapat menyelesaikan proses dengan algoritma MVCC sebagaimana mekanisme yang telah dijelaskan sebelumnya. Algoritma ini mengecek validitas timestamp ketika ingin melakukan read dan write pada objek dari suatu transaksi. Apabila transaksi memiliki timestamp yang lebih awal dari read timestamp objek saat ini ketika ingin menulis ke suatu objek tersebut, maka transaksi dibatalkan dan dimulai ulang dengan timestamp terbaru. Dari seluruh tes yang dilakukan, seluruh testcase diselesaikan dengan tepat.

# Eksplorasi Recovery

## Write-Ahead Log

Write-Ahead Logging (WAL) adalah metode standar untuk memastikan integritas data. Deskripsi terperinci dapat ditemukan di sebagian besar pemrosesan transaksi. Secara singkat, konsep utama WAL adalah perubahan pada file data (di mana tabel dan indeks berada) harus ditulis hanya setelah perubahan tersebut dicatat, yaitu, setelah catatan log yang menjelaskan perubahan telah dipindahkan ke penyimpanan permanen.

### Contoh simulasi Write-ahead Log

- Sebelum kegagalan:

```
Jun 03 18:51:41 linux1 systemd[1]: Starting PostgreSQL 12 database server...
Jun 03 18:51:41 linux1 postmaster[11118]: 2020-06-03 18:51:41.889 IST [11118] LOG:  starting PostgreSQL 12.3 on x86_64-pc...64-bit
Jun 03 18:51:41 linux1 postmaster[11118]: 2020-06-03 18:51:41.890 IST [11118] LOG:  listening on IPv4 address "192.168.1....t 5432"
Jun 03 18:51:41 linux1 postmaster[11118]: 2020-06-03 18:51:41.893 IST [11118] LOG:  listening on Unix socket "/var/run/po...5432"
Jun 03 18:51:41 linux1 postmaster[11118]: 2020-06-03 18:51:41.902 IST [11118] LOG:  redirecting log output to logging col...rocess
Jun 03 18:51:41 linux1 postmaster[11118]: 2020-06-03 18:51:41.947 IST [11118] HINT:  Future log output will appear in dir..."log".
Jun 03 18:51:41 linux1 systemd[1]: Started PostgreSQL 12 database server.
Hint: Some lines were ellipsized, use -l to show in full.
bash-4.2$ psql
psql (12.3)
Type "help" for help.
```

- Saat terjadi kegagalan :

```
bash-4.2$ cat postgresql.conf|grep wal_keep_segments
cat: postgresql.conf: No such file or directory
bash-4.2$ cat postgresql.conf|grep -i wal_keep_segments
cat: postgresql.conf: No such file or directory
bash-4.2$
```

- Setelah kegagalan :

```
bash-4.2$ cat postgresql.conf|grep -i wal_keep_segments
#wal_keep_segments = 0          # in logfile segments; 0 disables
bash-4.2$
bash-4.2$ 
bash-4.2$ cat postgresql.conf|grep -i max_wal_size
max_wal_size = 1GB
bash-4.2$ cat postgresql.conf|grep -i archive_command
#archive_command = ''           # command to use to archive a logfile segment
bash-4.2$
```

- Solusi recovery :

```
postgres=# alter system set max_wal_size = '1 GB';
ALTER SYSTEM
postgres=# alter system set wal_keep_segments = 10;
ALTER SYSTEM
postgres=# alter system set archive_mode = on;
ALTER SYSTEM
postgres=# alter system set archive_command = 'cp %p /postgres/logs/archive_wal%f';
ALTER SYSTEM
postgres=# alter system set archive_timeout= '1 h';
ALTER SYSTEM
postgres=#
postgres=# \q
bash-4.2$ sudo systemctl restart postgresql-12
```

```

# Copy the file to a safe location (like a mounted NFS volume)
archive_command = 'cp %p /mnt/nfs/%f'

# Not overwriting files is a good practice
archive_command = 'test ! -f /mnt/nfs/%f && cp %p /mnt/nfs/%f'

# Copy to S3 bucket
archive_command = 's3cmd put %p s3://BUCKET/path/%f'

# Copy to Google Cloud bucket
archive_command = 'gsutil cp %p gs://BUCKET/path/%f'

# An external script
archive_command = '/opt/scripts/archive_wal %p'

```

## Continuous Archiving

Sistem statistik kumulatif PostgreSQL mendukung pengumpulan dan pelaporan informasi tentang aktivitas server. Saat ini, akses ke tabel dan indeks dalam istilah blok disk dan baris individual dihitung. Jumlah total baris di setiap tabel, dan informasi tentang tindakan vakum dan analisis untuk setiap tabel juga dihitung. Jika diaktifkan, panggilan ke fungsi yang ditentukan pengguna dan total waktu yang dihabiskan di setiap fungsi juga akan dihitung. PostgreSQL juga mendukung pelaporan informasi dinamis tentang apa yang sebenarnya terjadi di sistem saat ini, seperti perintah persis yang sedang dijalankan oleh proses server lain, dan koneksi lain mana yang ada di sistem. Fasilitas ini tidak tergantung pada sistem statistik kumulatif.

### Contoh Simulasi Continuous Archiving

- Sebelum terjadi kegagalan :

```

#wal_writer_delay = 200ms          # (change requires restart)
#wal_writer_flush_after = 1MB      # 1-10000 milliseconds
#wal_skip_threshold = 2MB          # measured in pages, 0 disables

#commit_delay = 0                 # range 0-100000, in microseconds
#commit_siblings = 5               # range 1-1000

# - Checkpoints -
#checkpoint_timeout = 5min        # range 30s-1d
max_wal_size = 1GB
min_wal_size = 80MB
#checkpoint_completion_target = 0.5 # checkpoint target duration, 0.0 - 1.0
#checkpoint_flush_after = 256kB    # measured in pages, 0 disables
#checkpoint_warning = 30s          # 0 disables

```

- Saat terjadi kegagalan :

```

# - Archiving -
archive_mode = on                # enables archiving; off, on, or always
                                  # (change requires restart)
archive_command = 'test ! -f /var/lib/pgsql/13/data/pg13_archive/%f && cp %p /var/lib/pgsql/13/data/pg13_archive/%f' command to use to archive a logfile segment
                                  # placeholders: %p = path of file to archive
                                  # %f = file name only
                                  # e.g. 'test ! -f /mnt/server/archivedir/%f && cp %p /mnt/server/archivedir/%f'
#archive_timeout = 0              # force a logfile segment switch after this
                                  # number of seconds; 0 disables

```

- Solusi Recovery :

```
# - Archive Recovery -
# These are only used in recovery mode.

#restore_command = ''          # command to use to restore an archived logfile segment
#                                # placeholders: %p = path of file to restore
#                                # %f = file name only
#                                # e.g. 'cp /mnt/server/archivedir/%f %p'
#                                # (change requires restart)
#archive_cleanup_command = ''    # command to execute at every restartpoint
#recovery_end_command = ''      # command to execute at completion of recovery
```

## Point-in-Time Recovery

PITR adalah kemampuan untuk restore atau mengembalikan keadaan *database cluster* sesuai *point-in-time* yang spesifik. Selanjutnya kita bisa melakukan restore archive sesuai file archive yang kita inginkan. Menggunakan file archive lebih efisien dibandingkan dengan melakukan dump.

### Contoh Simulasi Point in Time Recovery

- Sebelum terjadi kegagalan :

```
[postgres@ip-172-31-42-215 pg_wal]$ sudo systemctl start postgresql-13
[postgres@ip-172-31-42-215 pg_wal]$ sudo systemctl status postgresql-13
● postgresql-13.service - PostgreSQL 13 database server
   Loaded: loaded (/usr/lib/systemd/system/postgresql-13.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2021-09-24 14:52:46 UTC; 7s ago
     Docs: https://www.postgresql.org/docs/13/static/
  Process: 1696 ExecStartPre=/usr/pgsql-13/bin/postgresql-13-check-db-dir ${PGDATA} (code=exited, status=0/SUCCESS)
 Main PID: 1701 (postmaster)
   Tasks: 9 (limit: 4787)
  Memory: 64.6M
 CGroup: /system.slice/postgresql-13.service
         └─1701 /usr/pgsql-13/bin/postmaster -D /var/lib/pgsql/13/data/
          ├─1703 postgres: logger
          ├─1707 postgres: checkpointer
          ├─1708 postgres: background writer
          ├─1710 postgres: stats collector
          ├─1715 postgres: walwriter
          ├─1716 postgres: autovacuum launcher
          ├─1717 postgres: archiver last was 000000010000000000000006.partial
          └─1718 postgres: logical replication launcher

Sep 24 14:52:46 ip-172-31-42-215.ap-south-1.compute.internal systemd[1]: Starting PostgreSQL 13 database server...
Sep 24 14:52:46 ip-172-31-42-215.ap-south-1.compute.internal postmaster[1701]: 2021-09-24 14:52:46.783 UTC [1701] LOG:  redirecting log output to logging directory
Sep 24 14:52:46 ip-172-31-42-215.ap-south-1.compute.internal postmaster[1701]: 2021-09-24 14:52:46.783 UTC [1701] HINT:  Future log output will appear in directory /var/log/postgresql/
Sep 24 14:52:46 ip-172-31-42-215.ap-south-1.compute.internal systemd[1]: Started PostgreSQL 13 database server.
```

- Saat terjadi kegagalan :

```
2021-09-24 14:37:58.486 UTC [882] LOG:  database system is shut down
2021-09-24 14:37:58.533 UTC [1522] LOG:  starting PostgreSQL 13.4 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 8.4.1 20200928 (Red Hat 8.4.1-1), 64-bit
2021-09-24 14:37:58.533 UTC [1522] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2021-09-24 14:37:58.533 UTC [1522] LOG:  listening on IPv6 address "::", port 5432
2021-09-24 14:37:58.534 UTC [1522] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2021-09-24 14:37:58.537 UTC [1522] LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
2021-09-24 14:37:58.540 UTC [1525] LOG:  database system was shut down at 2021-09-24 14:37:58 UTC
2021-09-24 14:37:58.543 UTC [1522] LOG:  database system is ready to accept connections
2021-09-24 14:42:05.312 UTC [1522] LOG:  received fast shutdown request
2021-09-24 14:42:05.319 UTC [1522] LOG:  aborting any active transactions
2021-09-24 14:42:05.322 UTC [1522] LOG:  background worker "logical replication launcher" (PID 1531) exited with exit code 1
2021-09-24 14:42:05.334 UTC [1522] LOG:  shutting down
2021-09-24 14:42:05.336 UTC [1522] LOG:  database system is shut down
2021-09-24 14:42:05.380 UTC [1567] LOG:  starting PostgreSQL 13.4 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 8.4.1 20200928 (Red Hat 8.4.1-1), 64-bit
2021-09-24 14:42:05.380 UTC [1567] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2021-09-24 14:42:05.382 UTC [1567] LOG:  listening on IPv6 address "::", port 5432
2021-09-24 14:42:05.384 UTC [1567] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2021-09-24 14:42:05.386 UTC [1570] LOG:  database system was shut down at 2021-09-24 14:42:05 UTC
2021-09-24 14:42:05.390 UTC [1567] LOG:  database system is ready to accept connections
2021-09-24 14:52:46.783 UTC [1701] LOG:  starting PostgreSQL 13.4 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 8.4.1 20200928 (Red Hat 8.4.1-1), 64-bit
2021-09-24 14:52:46.783 UTC [1701] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2021-09-24 14:52:46.783 UTC [1701] LOG:  listening on IPv6 address "::", port 5432
2021-09-24 14:52:46.785 UTC [1701] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2021-09-24 14:52:46.787 UTC [1701] LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
2021-09-24 14:52:46.790 UTC [1704] LOG:  database system was interrupted; last known up at 2021-09-24 14:44:10 UTC
cp: cannot stat '/var/lib/pgsql/13/data/pgl3_archive/00000002.history': No such file or directory
```

- Solusi Recovery :

```
cp: cannot stat '/var/lib/pgsql/13/data/pg13_archive/00000002.history': No such file or directory
2021-09-24 14:52:46.819 UTC [1704] LOG:  starting archive recovery
cp: cannot stat '/var/lib/pgsql/13/data/pg13_archive/000000010000000000000004': No such file or directory
2021-09-24 14:52:46.824 UTC [1704] LOG:  redo starts at 0/4000028
2021-09-24 14:52:46.825 UTC [1704] LOG:  consistent recovery state reached at 0/4000130
2021-09-24 14:52:46.826 UTC [1701] LOG:  database system is ready to accept read only connections
cp: cannot stat '/var/lib/pgsql/13/data/pg13_archive/000000010000000000000005': No such file or directory
cp: cannot stat '/var/lib/pgsql/13/data/pg13_archive/000000010000000000000006': No such file or directory
2021-09-24 14:52:46.831 UTC [1704] LOG:  invalid record length at 0/60000A0: wanted 24, got 0
2021-09-24 14:52:46.831 UTC [1704] LOG:  redo done at 0/6000028
cp: cannot stat '/var/lib/pgsql/13/data/pg13_archive/000000010000000000000006': No such file or directory
cp: cannot stat '/var/lib/pgsql/13/data/pg13_archive/00000002.history': No such file or directory
2021-09-24 14:52:46.840 UTC [1704] LOG:  selected new timeline ID: 2
2021-09-24 14:52:46.913 UTC [1704] LOG:  archive recovery complete
```

# Kesimpulan dan Saran

## Kesimpulan

Dalam menjalankan sistem manajemen database diperlukan adanya konsistensi, integritas, dan ketersediaan terhadap database. Setelah menjalankan sebuah operasi, harus dijamin integritas dan konsistensi data. Namun, menjalankan program secara serial dapat memakan waktu yang lama. Oleh karena itu diperlukan sistem *concurrency control*.

Dalam menjalankan operasi konkuren, diperlukan adanya *serializability*. *Serializability* merupakan sistem yang menjamin operasi konkuren yang dilakukan memiliki jadwal sebagaimana jadwal eksekusi yang serial. Selain itu, ada repeatable read yang merupakan sistem pembacaan terhadap data yang belum dicommitt. Ada juga read commit dan uncommitted read, yang menjadi mekanisme terhadap pembacaan sebuah data.

Dari sekian algoritma *concurrency control*, pada makalah ini diimplementasikan tiga jenis algoritme, yakni simple locking, optimistic concurrency control, dan multiversion timestamp concurrency control. Ketiga algoritma ini digunakan untuk menjadikan prinsip ACID pada operasi konkuren database.

Untuk menanggulangi error atau kerusakan data, sistem manajemen database memiliki mekanisme yang disebut *recovery system*. Terdapat beberapa metode pengembalian sistem, di antaranya write-ahead log (perubahan hanya boleh terjadi setelah adanya perubahan log dari data), *continuous archiving* (sistem pengumpulan dan pelaporan aktivitas server secara kontigu), dan *point-in-Time recovery* (kemampuan restore cluster database berdasarkan *specific point-in-Time* melalui file yang spesifik).

## Saran

Terdapat berbagai mekanisme untuk menunjang prinsip-prinsip keamanan, ketersediaan dan integritas database. Setiap mekanisme memiliki tujuan dan metodenya sendiri. Tidak terdapat *fit for all method*. Pemilihan metode yang efisien perlu mempertimbangkan tujuan, kondisi hardware dan kebutuhan sistem.

## **Pembagian Kerja**

<b>NIM - Nama</b>	<b>Pembagian</b>
10022124 - Hasbi Al Farabi	Eksplorasi Recovery
13520043 - Muhammad Risqi Firdaus	OCC, Kesimpulan & Saran
13520133 - Jevant Jedia Augustine	Eksplorasi Concurrency Control, Simple Locking
13520160 - Willy Wilsen	Eksplorasi Concurrency Control, MVCC

## **Referensi**

Silberschatz, A., Korth, H. F., Sudarshan, S. (2010). *Database System Concepts*. New York: McGraw-Hill.