

**IF3140 MANAJEMEN BASIS DATA**

**MEKANISME CONCURRENCY CONTROL DAN RECOVERY**

**EmBeDeOke2**



**Kelompok 07 K02**

Anggota :

Fadil Fauzani	13520032
Shadiq Harwiz	13520038
Alifia Rahmah	13520122
Febryola Kurnia Putri	13520140

**Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika**

**Institut Teknologi Bandung**

**2022**

# Daftar Isi

<b>Daftar Isi</b>	<b>1</b>
<b>1. Hasil Eksplorasi Concurrency Control</b>	<b>2</b>
a. Serializable	2
b. Repeatable Read	3
c. Read Committed	4
d. Read Uncommitted	5
<b>2. Implementasi Concurrency Control Protocol</b>	<b>6</b>
a. Simple Locking (exclusive locks only)	6
b. Serial Optimistic Concurrency Control (OCC)	19
<b>3. Hasil Eksplorasi Recovery</b>	<b>29</b>
a. Write-Ahead Log	29
b. Continuous Archiving	29
c. Point-in-Time Recovery	29
d. Simulasi Kegagalan pada PostgreSQL	30
Set-up PostgreSQL untuk mendukung recovery	30
Backup dan Simulasi kegagalan	31
Melakukan recovery	33
<b>4. Kesimpulan dan Saran</b>	<b>36</b>
a. Kesimpulan	36
b. Saran	36
<b>5. Pembagian Kerja</b>	<b>38</b>
<b>Referensi</b>	<b>39</b>

# 1. Hasil Eksplorasi Concurrency Control

Pada PostgreSQL, Kita bisa membuat Transaksi dengan *keyword* “BEGIN” lalu menuliskan semua task pada Transaksi dan menuliskan *keyword* “Commit” pada akhir transaksi. Kita bisa menspesifikan derajat isolasi pada transaksi yang kita buat dengan *keyword* “BEGIN TRANSACTION ISOLATION LEVEL \_” dengan tanda strip ( ) diganti dengan jenis derajat isolasi yang kita mau. Terdapat 3 jenis derajat isolasi transaksi pada PostgreSQL, yaitu *Serializable*, *Repeatable Read*, dan *Read Committed* dengan masing masing transaksi memiliki karakteristiknya sendiri. Jika kita tidak spesifikkan jenis isolasi transaksi yang kita buat, transaksi tersebut otomatis memiliki jenis isolasi *Read Committed*.

## a. Serializable

Pada derajat isolasi *Serializable*, data dapat dilihat pada transaksi adalah data yang telah di-commit sebelum transaksi dimulai dan PostgreSQL hanya menerima urutan transaksi yang *serializable*. Jika terdapat query pada salah satu transaksi yang menyebabkan eksekusinya menjadi tidak *serializable*, PostgreSQL akan memberikan error berupa “*ERROR: could not serialize access due to ...*”. Akibatnya, pada derajat isolasi ini, memiliki derajat konsistensi paling tinggi namun derajat serializability kecil.

```
binotify=# BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN
binotify=# select sum(duration) from song where album_id = 2;
sum
-----
866
(1 row)

binotify=# INSERT INTO song(judul, penyanyi, tanggal_terbit, genre, duration, audio_path, image_path, album_id) VALUES ('Lupakan Reyhan', 'Intan', now(), 'House', 199, 'resources/music/audio/Dim Yosef - Canary [ACS Release].mp3', 'resources/music/image/song/canary.jpg', 2);
INSERT 0 1
binotify=# commit;
COMMIT
binotify=# |

binotify=# BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN
binotify=# select sum(duration) from song where album_id = 1;
sum
-----
866
(1 row)

binotify=# INSERT INTO song(judul, penyanyi, tanggal_terbit, genre, duration, audio_path, image_path, album_id) VALUES ('Kini Echo Pergi Meninggalkanku', 'Intan', now(), 'House', 199, 'resources/music/audio/Dim Yosef - Canary [ACS Release].mp3', 'resources/music/image/song/canary.jpg', 1);
INSERT 0 1
binotify=# commit;
ERROR: could not serialize access due to read/write dependencies among transactions
DETAIL: Reason code: Canceled on identification as a pivot, during commit attempt.
HINT: The transaction might succeed if retried.
binotify=# |
```

Gambar 1.1 Hasil Insert kedua transaksi dengan isolasi Serializable

Terlihat pada kedua gambar, karena kedua transaksi tersebut tidak serializable, salah satu transaksi tidak bisa commit, berbeda dengan pada isolasi Repeatable read

## b. Repeatable Read

Pada derajat isolasi *Repeatable Read*, data dapat dilihat pada transaksi adalah data yang telah di-*commit* sebelum transaksi dimulai. Bedanya dengan isolasi *serializable*, Repeatable Read masih menerima urutan transaksi yang tidak serializable jika tidak melakukan update, PostgreSQL akan memberikan error jika terdapat *concurrent update* berupa “*ERROR: could not serialize access due to ...*”. Akibatnya, pada derajat isolasi ini bisa saja terjadi Serialization anomaly dimana urutan *insert* hasilnya berbeda dengan jika melakukan Transaksi secara serial.

```
binotify=# BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN
binotify=# INSERT INTO song(judul, penyanyi, tanggal_terbit, genre, duration, audio_path, image_path, album_id) VALUES ('Lupakan Reyhan', 'Intan', now(), 'House', 199, 'resources/music/audio/31m Yosef - Canary [NCS Release].mp3', 'resources/music/image/song/canary.jpg', 1);
INSERT 0 1
binotify=# commit;
```

Gambar 1.2 Insert transaksi 1 percobaan 1

```
binotify=# BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN
binotify=# INSERT INTO song(judul, penyanyi, tanggal_terbit, genre, duration, audio_path, image_path, album_id) VALUES ('Kini Echo Pergi Meninggalkanku', 'Intan', now(), 'House', 199, 'resources/music/audio/31m Yosef - Canary [NCS Release].mp3', 'resources/music/image/song/canary.jpg', 1);
INSERT 0 1
binotify=# commit;
COMMIT
```

Gambar 1.3 Insert transaksi 2 percobaan 1

song_id	judul	penyanyi	tanggal_terbit	genre	duration	audio_path
	image_path	album_id				
1	Canary	Jim Yosef	2016-10-29	House	199	resources/music/audio/31m Yosef - Canary [NCS Release].mp3
2	Safe and Sound	Different Heaven	2016-12-16	Dance	205	resources/music/audio/Different Heaven - Safe And Sound [NCS Release].mp3
3	Firefly pt. II	Jim Yosef	2020-12-31	House	225	resources/music/audio/31m Yosef - Firefly pt. II (ft. STARLYTE) [NCS Release].mp3
4	Linked	Jim Yosef & Anna Yvette	2017-12-15	House	203	resources/music/audio/31m Yosef & Anna Yvette - Linked [NCS Release].mp3
5	Dreams	Lost Sky	2018-07-25	Dance	215	resources/music/audio/Lost Sky - Dreams [NCS Release].mp3
6	Need You	Lost Sky	2020-12-25	Dance	277	resources/music/audio/Lost Sky - Need You [NCS Release].mp3
7	Where We Started (Feat. Jex)	Lost Sky	2019-11-08	Dance	222	resources/music/audio/Lost Sky - Where We Started (Feat. Jex) [NCS Release].mp3
8	Dancing On The Moon (ft. Luke Burr)	Unknown Brain	2020-09-04	Electronic	228	resources/music/audio/Unknown Brain - Dancing On The Moon (ft. Luke Burr) [NCS Release].mp3
9	Perfect 10 (feat. Heather Sommer)	Unknown Brain	2019-04-26	Electronic	204	resources/music/audio/Unknown Brain - Perfect 10 (feat. Heather Sommer) [NCS Release].mp3
10	Why do I (feat. Bri Tolani)	Unknown Brain	2018-04-17	Electronic	223	resources/music/audio/Unknown Brain - Why Do I. (feat. Bri Tolani) [NCS Release].mp3
11	Lupakan Reyhan	Intan	2022-12-04	House	199	resources/music/audio/31m Yosef - Canary [NCS Release].mp3
12	Kini Echo Pergi Meninggalkanku	Intan	2022-12-04	House	199	resources/music/audio/31m Yosef - Canary [NCS Release].mp3
(12 rows)						

Gambar 1.4 Hasil Insert kedua transaksi dengan isolasi repeatable read percobaan 1

```
binotify=# BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN
binotify=# INSERT INTO song(judul, penyanyi, tanggal_terbit, genre, duration, audio_path, image_path, album_id) VALUES ('Lupakan Reyhan', 'Intan', now(), 'House', 199, 'resources/music/audio/31m Yosef - Canary [NCS Release].mp3', 'resources/music/image/song/canary.jpg', 1);
INSERT 0 1
binotify=# commit;
```

Gambar 1.5 Insert transaksi 1 percobaan 1

```
binotify=# BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN
binotify=# INSERT INTO song(judul, penyanyi, tanggal_terbit, genre, duration, audio_path, image_path, album_id) VALUES ('Kini Echo Pergi Meninggalkanku', 'Intan', now(), 'House', 199, 'resources/music/audio/31m Yosef - Canary [NCS Release].mp3', 'resources/music/image/song/canary.jpg', 1);
INSERT 0 1
binotify=# commit;
COMMIT
```

Gambar 1.6 Insert transaksi 2 percobaan 1

song_id	judul	penyanyi	tanggal_terbit	genre	duration	audio_path
	image_path	album_id				
1	Canary	Jim Yosef	2016-10-29	House	199	resources/music/audio/31m Yosef - Canary [NCS Release].mp3
2	Safe and Sound	Different Heaven	2016-12-16	Dance	205	resources/music/audio/Different Heaven - Safe And Sound [NCS Release].mp3
3	Firefly pt. II	Jim Yosef	2020-12-31	House	225	resources/music/audio/31m Yosef - Firefly pt. II (ft. STARLYTE) [NCS Release].mp3
4	Linked	Jim Yosef & Anna Yvette	2017-12-15	House	203	resources/music/audio/31m Yosef & Anna Yvette - Linked [NCS Release].mp3
5	Dreams	Lost Sky	2018-07-25	Dance	215	resources/music/audio/Lost Sky - Dreams [NCS Release].mp3
6	Need You	Lost Sky	2020-12-25	Dance	277	resources/music/audio/Lost Sky - Need You [NCS Release].mp3
7	Where We Started (Feat. Jex)	Lost Sky	2019-11-08	Dance	222	resources/music/audio/Lost Sky - Where We Started (Feat. Jex) [NCS Release].mp3
8	Dancing On The Moon (ft. Luke Burr)	Unknown Brain	2020-09-04	Electronic	228	resources/music/audio/Unknown Brain - Dancing On The Moon (ft. Luke Burr) [NCS Release].mp3
9	Perfect 10 (feat. Heather Sommer)	Unknown Brain	2019-04-26	Electronic	204	resources/music/audio/Unknown Brain - Perfect 10 (feat. Heather Sommer) [NCS Release].mp3
10	Why do I (feat. Bri Tolani)	Unknown Brain	2018-04-17	Electronic	223	resources/music/audio/Unknown Brain - Why Do I. (feat. Bri Tolani) [NCS Release].mp3
11	Lupakan Reyhan	Intan	2022-12-04	House	199	resources/music/audio/31m Yosef - Canary [NCS Release].mp3
12	Kini Echo Pergi Meninggalkanku	Intan	2022-12-04	House	199	resources/music/audio/31m Yosef - Canary [NCS Release].mp3
(12 rows)						

Gambar 1.7 Hasil Insert kedua transaksi dengan isolasi repeatable read percobaan 1

Terlihat pada kedua gambar, urutan kedua hasil transaksi tersebut berbeda bergantung dengan query insert mana yang dipanggil duluan walaupun transaksi 1 merupakan transaksi yang paling dahulu datang. Ini akan menyebabkan masalah pada konsistensi

### c. Read Committed

Pada derajat isolasi *Read Committed*, data dapat dilihat pada *query* pada transaksi adalah data yang telah di-*commit* sebelum *query* pada transaksi tersebut. Akibatnya, pada derajat isolasi ini bisa muncul masalah *phantom* dan *non-repeatable read* karena hasil *query* bisa saja berubah ketika transaksi lain melakukan *commit*.

```
binotify=# BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN
binotify=# select * from song where song_id = 1;
 song_id | judul | penyanyi | tanggal_terbit | genre | duration | audio_path
-----+-----+-----+-----+-----+-----+-----
1 | UAS sudah tiba | Jim Yosef | 2016-10-29 | House | 199 | resources/music/audio/Jim Yosef - Canary [NCS Release].mp3 | resources/music/image/song/canary.jpg | 1
(1 row)

binotify=# select * from song where song_id = 1;
 song_id | judul | penyanyi | tanggal_terbit | genre | duration | audio_path
-----+-----+-----+-----+-----+-----+-----
1 | BESOK UAS | Jim Yosef | 2016-10-29 | House | 199 | resources/music/audio/Jim Yosef - Canary [NCS Release].mp3 | resources/music/image/song/canary.jpg | 1
(1 row)

binotify=# commit;
```

Gambar 1.8 Simulasi masalah *non-repeatable read* pada *Read Committed*

```
binotify=# BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
BEGIN
binotify=# select * from song where judul like 'KS';
 song_id | judul | penyanyi | tanggal_terbit | genre | duration | audio_path | image_path | album_id
-----+-----+-----+-----+-----+-----+-----+-----+-----
1 | BESOK UAS | Jim Yosef | 2016-10-29 | House | 199 | resources/music/audio/Jim Yosef - Canary [NCS Release].mp3 | resources/music/image/song/canary.jpg | 1
(1 row)

binotify=# select * from song where judul like 'KS';
 song_id | judul | penyanyi | tanggal_terbit | genre | duration | audio_path | image_path | album_id
-----+-----+-----+-----+-----+-----+-----+-----+-----
1 | BESOK UAS | Jim Yosef | 2016-10-29 | House | 199 | resources/music/audio/Jim Yosef - Canary [NCS Release].mp3 | resources/music/image/song/canary.jpg | 1
(1 row)

binotify=# select * from song where judul like 'KS';
 song_id | judul | penyanyi | tanggal_terbit | genre | duration | audio_path | image_path | album_id
-----+-----+-----+-----+-----+-----+-----+-----+-----
1 | BESOK UAS | Jim Yosef | 2016-10-29 | House | 199 | resources/music/audio/Jim Yosef - Canary [NCS Release].mp3 | resources/music/image/song/canary.jpg | 1
11 | malaS | kuFaku | 2022-12-04 | dangdut | 199 | ./ | ./ | 1
(2 rows)
```

Gambar 1.9 Simulasi masalah *phantom read* pada *Read Committed*

Terlihat pada kedua gambar, pada *Read Committed* nilai suatu *query* bisa saja berubah ini akan menyebabkan masalah pada konsistensi.

### d. Read Uncommitted

PostgreSQL tidak mengimplementasikan derajat isolasi *Read Uncommitted*, derajat isolasi yang paling bawah adalah *Read committed (default)*. Namun pada konsepnya, tiap *query* pada transaksi *read uncommitted* bisa melakukan read pada tiap perubahan data (walaupun belum di-*commit*). Akibatnya pada *Read uncommitted* akan terjadi *dirty read* dimana transaksi melakukan read pada nilai yang belum *commit*.

## 2. Implementasi Concurrency Control Protocol

### a. Simple Locking (exclusive locks only)

*Lock* adalah sebuah mekanisme untuk mengontrol akses konkuren dari sebuah item data. Permintaan *lock* terlebih dahulu dikirim ke *concurrency control manager*. Jika *lock* mendapat izin, maka transaksi dapat melakukan akses terhadap item data. Terdapat dua jenis *lock*, yaitu *exclusive mode* dan *share mode*. Pada *exclusive mode*, transaksi dapat melakukan pembacaan dan/atau penulisan terhadap item data. Pada *share mode*, transaksi hanya dapat melakukan pembacaan terhadap item data. Hal ini dikarenakan *share mode* memberikan akses *lock* yang dapat digunakan oleh transaksi yang lain.

Berikut implementasi kode program pada *simple locking (exclusive locks only)* dengan menggunakan bahasa pemrograman python.

1. **Kelas Transaction** untuk menginisialisasi semua variabel awal yang diperlukan

```
class Transaction:
    def __init__(self, id):
        self.id = id
        self.listTransaction = []
        self.listAllTransaction = []
        self.waitTransactionExclusiveLockFrom = ""
        self.indexwaitTransactionExclusiveLockFrom = ""
```

2. **Kelas Exclusive Lock**, untuk menginisialisasi semua variabel awal yang diperlukan

```
class Transaction:
    def __init__(self, id):
        self.id = id
        self.listTransaction = []
        self.listAllTransaction = []
        self.waitTransactionExclusiveLockFrom = ""
        self.indexwaitTransactionExclusiveLockFrom = ""
```

3. **Proses Validasi OCC** yang diperlukan untuk melakukan validasi terhadap schedule yang ada apakah pada transaksi tertentu sudah mencapai tahap commit dan tidak ada data yang belum dieksekusi

```
def validasiTransaksi(transaction) -> bool:
    if ( ( len(transaction.listTransaction) == 1 ) and ( transaction.listTransaction[0][0] == "C" ) ) :
        return True
```

4. **Get Index List Exclusive Lock** yang diperlukan untuk mendapatkan index pada class exclusive lock

```
def getIndexListExclusiveLock(data, listExclusiveLock):
    for i in range (len(listExclusiveLock)) :
        if ( listExclusiveLock[i].data == data) :
            return (i)
```

5. **Exclusive Lock Release** yang diperlukan untuk melepaskan exclusive pada data tertentu yang sedang dipegang oleh transaksi tertentu

```
def exclusiveLockRelease(idTransaction, listExclusiveLock, listUrutanSchedule) :
    for i in range (len(listExclusiveLock)) :
        if (listExclusiveLock[i].whoTransactionGetExlusiveLock == idTransaction) :
            listUrutanSchedule.append("UL" + idTransaction + "(" + listExclusiveLock[i].data + ")")
            print("==> Exclusive Lock " + listExclusiveLock[i].data + " on transaction " + idTransaction + " release")
            listExclusiveLock[i].whoTransactionGetExlusiveLock = ""
    return (listUrutanSchedule)
```

6. **Move Queue To Schedule** yang diperlukan untuk memindahkan transaksi yang berada pada antrian ke dalam schedule

```
def moveQueueToSchedule(queueOperasi, listSchedule, listAbortTransaction) :
    tempListSchedule = []
    for i in range (len(queueOperasi)):
        if (queueOperasi[i] not in listAbortTransaction) :
            tempListSchedule.append(queueOperasi[i])
    for i in range (len(listSchedule)) :
        if (listSchedule[i] not in listAbortTransaction) :
            tempListSchedule.append(listSchedule[i])
    del listSchedule
    del queueOperasi
    listSchedule = []
    queueOperasi = []
    for i in range (len(tempListSchedule)) :
        listSchedule.append(tempListSchedule[i])
    del tempListSchedule
    return(queueOperasi, listSchedule)
```

7. **Elimination Transaction Abort** yang diperlukan untuk mengeliminasi seluruh data yang berkaitan dengan transaksi tertentu pada schedule dan antrian



```
def eliminationTransactionAbort(listSchedule, listAbortTransaction):
    tempListSchedule = []
    for j in range (len(listSchedule)) :
        if (listSchedule[j] not in listAbortTransaction):
            tempListSchedule.append(listSchedule[j])
    del listSchedule
    listSchedule = []
    for j in range (len(tempListSchedule)) :
        listSchedule.append(tempListSchedule[j])
    del tempListSchedule
    return(listSchedule)
```

8. **Check Exclusive Lock** yang diperlukan untuk melakukan pengecekan pada suatu transaksi tertentu dengan data tertentu apakah transaksi tersebut dapat mengambil exclusive lock pada data tersebut atau tidak. Jika transaksi tersebut tidak mendapat exclusive lock, transaksi tersebut akan dipindahkan ke dalam antrian dan kemudian jika menerapkan wound-wait scheme, maka akan dilakukan pengecekan apakah transaksi yang meminta exclusive lock merupakan transaksi yang lebih tua daripada transaksi yang sedang memegang exclusive lock

```
def checkExclusiveLock(listExclusiveLock, indexExclusiveLock, listTransaksi, id, idStr, listSchedule, queueOperasi, data, listQueueData, listQueueTransaction, listUrutanSchedule, isUseWoundWaitSchema, listUrutanTransaksi, listAbortTransaction, listofListAbortTransaction) :
    if listExclusiveLock[indexExclusiveLock].whoTransactionGetExclusiveLock == "" and listTransaksi[id].listTransaction[0] == listSchedule[0] :
        if ( listTransaksi[id].waitTransactionExclusiveLockFrom != "" ) :
            listTransaksi[id].waitTransactionExclusiveLockFrom = ""
            listTransaksi[id].indexwaitTransactionExclusiveLockFrom = ""
        listExclusiveLock[indexExclusiveLock].whoTransactionGetExclusiveLock = idStr
        print("==> Give Exclusive Lock " + data + " to T" + idStr)
        listUrutanSchedule.append("Xl" + idStr + "(" + data + ")")
        listUrutanSchedule.append(listSchedule[0])
        del listSchedule[0]
        del listTransaksi[id].listTransaction[0]
        print("==> Success")
    elif listExclusiveLock[indexExclusiveLock].whoTransactionGetExclusiveLock == idStr and listTransaksi[id].listTransaction[0] == listSchedule[0] :
        if ( listTransaksi[id].waitTransactionExclusiveLockFrom != "" ) :
            listTransaksi[id].waitTransactionExclusiveLockFrom = ""
            listTransaksi[id].indexwaitTransactionExclusiveLockFrom = ""
        listUrutanSchedule.append(listSchedule[0])
        del listSchedule[0]
        del listTransaksi[id].listTransaction[0]
        print("==> Success")
    else :
        if ( listTransaksi[id].waitTransactionExclusiveLockFrom == "" ) :
            listTransaksi[id].waitTransactionExclusiveLockFrom = data
            listTransaksi[id].indexwaitTransactionExclusiveLockFrom = listExclusiveLock[indexExclusiveLock].whoTransactionGetExclusiveLock
        print("==> Transaction " + idStr + " is waiting for Exclusive Lock " + listTransaksi[id].waitTransactionExclusiveLockFrom + " which is in Transaction " + listTransaksi[id].indexwaitTransactionExclusiveLockFrom + ". " + listSchedule[0] + " get into the queue...")
        queueOperasi.append(listSchedule[0])
        if ( data not in listQueueData ) :
            listQueueData.append(data)
        if ( idStr not in listQueueTransaction ) :
            listQueueTransaction.append(idStr)
        del listSchedule[0]
```

```

# Cek apakah transaksi tua meminta lock kepada transaksi yang muda dari nya
isOldAskYoung = False
susOld = idStr
urutanSusOld = 0
susYoung = ""
urutanSusYoung = 0
for i in range (len(listExclusiveLock)) :
    if (listExclusiveLock[i].data == listTransaksi[id].waitTransactionExclusiveLockFrom) :
        susYoung = listExclusiveLock[i].whoTransactionGetExclusiveLock

for i in range (len(listUrutanTransaksi)) :
    if (str(listUrutanTransaksi[i]) == susOld):
        urutanSusOld = i
    if (str(listUrutanTransaksi[i]) == susYoung):
        urutanSusYoung = i

if (urutanSusOld < urutanSusYoung) :
    isOldAskYoung = True
else :
    isOldAskYoung = False

if (isUseWoundWaitSchema == True and isOldAskYoung == True ) :
    if ( len(listAbortTransaction) != 0 ) :
        listofListAbortTransaction.append(listAbortTransaction)
        del listAbortTransaction
        listAbortTransaction = []

    print("!!!Do Wound-Wait Scheme!!!")
    print("Aborting T", end="")
    idTransactionPrevent = int(susYoung)
    id = idTransactionPrevent - 1
    listUrutanSchedule.append("A" + str(idTransactionPrevent))

#Transaksi yang di-abort masuk ke dalam listAbortTransaction
for j in range (len(listTransaksi[id].listAllTransaction)):
    listAbortTransaction.append(listTransaksi[id].listAllTransaction[j])

```

```

# Fase pembebasan exclusive lock yang ada pada transaksi tersebut
for j in range ( len(listExclusiveLock) ):
    if (listExclusiveLock[j].whoTransactionGetExclusiveLock == str(idTransactionPrevent)) :
        listUrutanSchedule.append("UL" + str(idTransactionPrevent) + "(" + listExclusiveLock[j].data + ")")
        print("=> Exclusive Lock " + listExclusiveLock[j].data + " on transaction " + str(idTransactionPrevent) + " release")
        listExclusiveLock[j].whoTransactionGetExclusiveLock = ""

# Fase eliminasi transaksi yang di-abort
if ( len(queueOperasi) > 0 ) :
    queueOperasi, listSchedule = moveQueueToSchedule(queueOperasi, listSchedule, listAbortTransaction)
else :
    listSchedule = eliminationTransactionAbort(listSchedule, listAbortTransaction)

# Fase penghapusan list
del listQueueData
listQueueData = []

```

9. **LockTransaction** yang diperlukan sebagai proses dari *simple locking (exclusive lock only)* bekerja

```

def LockTransaction(listSchedule, totalTransaksi, isUseWoundWaitSchema):
    print(Fore.BLUE+"-----"+Fore.RESET)
    print(Fore.RED+"-----Simple Locking-----"+Fore.RESET)
    print(Fore.BLUE+"-----"+Fore.RESET)

    # Fase Inisialisasi
    # Inisialisasi kamus
    isDeadLock = False
    queueOperasi = []
    listExclusiveLock = []
    listData = []
    listTransaksi = []
    listQueueData = []
    listUrutanTransaksi = []
    listUrutanSchedule = []
    listQueueTransaction = []
    listAbortTransaction = []
    listOfListAbortTransaction = []

    # Inisialisasi list transaksi
    for i in range (totalTransaksi):
        transaksi = Transaction(i+1)
        listTransaksi.append(transaksi)

    # Inisialisasi list data yang akan ditransaksi
    print("Berikut Urutan Transaksi")
    for i in range (len(listSchedule)) :
        idTransaksi = 0

        #Kalau schedulanya berupa commit
        if (len(listSchedule[i]) == 2) :
            idTransaksi = int(listSchedule[i][1]) - 1
        elif ( len(listSchedule[i]) == 3 ) :
            idTransaksi = (int( listSchedule[i][1] + listSchedule[i][2] )) - 1

```

```

#Kalau schedulanya berupa write dan read
if ( len(listSchedule[i]) == 5 ) :
    idTransaksi = int(listSchedule[i][1]) - 1

    if listSchedule[i][3] not in listData :
        exclusiveClock = ExclusivceClock(listSchedule[i][3])
        listExclusiveLock.append(exclusiveClock)
        listData.append(listSchedule[i][3])

    if ((idTransaksi + 1) not in listUrutanTransaksi):
        listUrutanTransaksi.append(idTransaksi + 1)
        print("Transaksi", idTransaksi + 1)
elif ( len(listSchedule[i]) == 6 ) :
    idTransaksi = (int( listSchedule[i][1] + listSchedule[i][2] )) - 1

    if listSchedule[i][4] not in listData :
        exclusiveClock = ExclusivceClock(listSchedule[i][4])
        listExclusiveLock.append(exclusiveClock)
        listData.append(listSchedule[i][4])

    if ((idTransaksi + 1) not in listUrutanTransaksi):
        listUrutanTransaksi.append(idTransaksi + 1)
        print("Transaksi", idTransaksi + 1)

(listTransaksi[idTransaksi].listTransaction).append(listSchedule[i])
(listTransaksi[idTransaksi].listAllTransaction).append(listSchedule[i])

print()
# Fase operasi
while ( len(listSchedule) > 0 ):
    i = 0

```

```

# Kasus commit
if (listSchedule[i][0] == "C"):
    idTransaction = 0
    idTransactionStr = ""

    print(Fore.BLUE+Back.WHITE+"Commit"+Fore.RESET + Back.RESET+" ", end="")
    print("transaksi T", end="")

    if ( len(listSchedule[i]) == 2 ):
        idTransaction = int(listSchedule[i][1]) - 1
        idTransactionStr = listSchedule[i][1]
        print(idTransactionStr)
    elif ( len(listSchedule[i]) == 3 ):
        idTransaction = int( listSchedule[i][1] + listSchedule[i][2] ) - 1
        idTransactionStr = (listSchedule[i][1] + listSchedule[i][2])
        print(idTransactionStr)

# Fase Validasi
hasilValidasi = validasiTransaksi(listTransaksi[idTransaction])

if (hasilValidasi == True):
    print("==> Success...")

    # Fase pembebasan exclusive lock yang ada pada transaksi tersebut
    listUrutanSchedule = exclusiveLockRelease(idTransactionStr, listExclusiveLock, listUrutanSchedule)

    # Fase penghapusan transaksi yang berhasil di-commit dari listTransaksi dan listSchedule
    del listTransaksi[idTransaction].listTransaction
    listUrutanSchedule.append(listSchedule[i])
    del listSchedule[i]

    # Fase pemindahan queueOperasi ke listSchedule
    if ( len(queueOperasi) > 0 ) :
        queueOperasi, listSchedule = moveQueueToSchedule(queueOperasi, listSchedule, listAbortTransaction)

```

```

# Fase penghapusan transaksi yang telah berhasil di-commit dari listUrutanTransaksi
for j in range (len(listUrutanTransaksi)) :
    if listUrutanTransaksi[j] == (idTransaction + 1) :
        del listUrutanTransaksi[j]
        break

else:
    print("==> Pending...")

# Fase pemindahan transaksi itu ke queueOperasi
queueOperasi.append(listSchedule[i])

# Hapus transaksi itu dari listSchedule
del listSchedule[i]

# Kasus Read atau Write
elif (listSchedule[i][0] == "R" or listSchedule[i][0] == "W"):

    if(len(listSchedule[i]) == 5):
        id = int( listSchedule[i][1] ) - 1
        idStr = listSchedule[i][1]
        data = listSchedule[i][3]
        indexExclusiveLock = getIndexListExclusiveLock(data, listExclusiveLock)

        if (listSchedule[i][0] == "R"):
            print(Fore.YELLOW+Back.WHITE+"Read"+Fore.RESET + Back.RESET+" ", end="")
        else: #(listSchedule[i][0] == "W")
            print(Fore.RED+Back.WHITE+"Write"+Fore.RESET + Back.RESET+" ", end="")

        print(data, "pada T", end="")
        print(idStr)

```

```

#Cek exclusive lock sedang dipegang transaksi apa dan apakah terdapat transaksi sebelum si transaksi tersebut
listExclusiveLock, listSchedule, listTransaksi, queueOperasi, listQueueData, listQueueTransaction, listUrutanSchedule, listUrutanTransaksi,
listAbortTransaction, listofListAbortTransaction = checkExclusiveLock(listExclusiveLock, indexExclusiveLock, listTransaksi, id, idStr,
listSchedule, queueOperasi, data, listQueueData, listQueueTransaction, listUrutanSchedule, isUseWoundWaitSchema, listUrutanTransaksi,
listAbortTransaction, listofListAbortTransaction)

elif(len(listSchedule[i]) == 6):
    id = (int( listSchedule[i][1] + listSchedule[i][2] )) - 1
    idStr = listSchedule[i][1] + "" + listSchedule[i][2]
    data = listSchedule[i][4]
    indexExclusiveLock = getIndexListExclusiveLock(data, listExclusiveLock)

    if (listSchedule[i][0] == "R"):
        print(Fore.YELLOW+Back.WHITE+"Read"+Fore.RESET + Back.RESET+" ", end="")
    else: #(listSchedule[i][0] == "W")
        print(Fore.RED+Back.WHITE+"Write"+Fore.RESET + Back.RESET+" ", end="")

    print(data,"pada T", end="")
    print(idStr)

#Cek exclusive lock sedang dipegang transaksi apa dan apakah terdapat transaksi sebelum si transaksi tersebut
listExclusiveLock, listSchedule, listTransaksi, queueOperasi, listQueueData, listQueueTransaction, listUrutanSchedule, listUrutanTransaksi,
listAbortTransaction, listofListAbortTransaction = checkExclusiveLock(listExclusiveLock, indexExclusiveLock, listTransaksi, id, idStr,
listSchedule, queueOperasi, data, listQueueData, listQueueTransaction, listUrutanSchedule, isUseWoundWaitSchema, listUrutanTransaksi,
listAbortTransaction, listofListAbortTransaction)

#Cek jika ternyata terjadi proses saling menunggu yang menyebabkan deadlock
if ( len(listData) == len(listQueueData) and len(listUrutanTransaksi) == len(listQueueTransaction) and len(listQueueData) > 1 and len
(listQueueTransaction) > 1 and len(listSchedule) != 0 and isUseWoundWaitSchema == False) :
    print("!!!Deadlock detected!!!")
    del listSchedule
    listSchedule = []
    isDeadlock = True
    break

```

```

#Fase menjalankan transaksi yang telah diabort
if ( len(listSchedule) == 0 and len(listAbortTransaction) != 0 ) :
    print("!!!Executing aborted transactions!!!")
    listOfListAbortTransaction.append(listAbortTransaction)
    if (len(listOfListAbortTransaction) > 0) :
        for i in range (len(listOfListAbortTransaction)) :
            del listAbortTransaction
            listAbortTransaction = []
            for j in range (len(listOfListAbortTransaction[i])):
                listAbortTransaction.append((listOfListAbortTransaction[i])[j])
            #Cari id transaksi yang di abort
            id = 0
            if ( len(listAbortTransaction[0]) == 5 or len(listAbortTransaction[0]) == 2):
                id = int(listAbortTransaction[0][1]) - 1
            elif ( len(listAbortTransaction[0]) == 6 or len(listAbortTransaction[0]) == 3):
                id = int(listAbortTransaction[0][1] + listAbortTransaction[0][2]) - 1
            del listTransaksi[id].listTransaction
            listTransaksi[id].listTransaction = []
            for i in range (len(listTransaksi[id].listAllTransaction)) :
                (listTransaksi[id].listTransaction).append(listTransaksi[id].listAllTransaction[i])
            for j in range (len(listAbortTransaction)) :
                listSchedule.append(listAbortTransaction[j])
            del listAbortTransaction
            listAbortTransaction = []

else :
    #Cari id transaksi yang di abort
    id = 0
    if ( len(listAbortTransaction[0]) == 5 or len(listAbortTransaction[0]) == 2):
        id = int(listAbortTransaction[0][1]) - 1
    elif ( len(listAbortTransaction[0]) == 6 or len(listAbortTransaction[0]) == 3):
        id = int(listAbortTransaction[0][1] + listAbortTransaction[0][2]) - 1
    del listTransaksi[id].listTransaction
    listTransaksi[id].listTransaction = []
    for i in range (len(listTransaksi[id].listAllTransaction)) :
        (listTransaksi[id].listTransaction).append(listTransaksi[id].listAllTransaction[i])
    for j in range (len(listAbortTransaction)) :
        listSchedule.append(listAbortTransaction[j])
    del listAbortTransaction
    listAbortTransaction = []

if (len(listSchedule) == 0 and len(queueOperasi) == 0 and isDeadLock == False) :
    print("\nBerikut Urutan Schedule")
    for i in range (len(listUrutanSchedule)) :
        print(listUrutanSchedule[i], end="")
        print("; ", end="")
    print("\n")
    return True
else :
    return False

```

Berikut testing dari kode program pada *simple locking (exclusive locks only)* dengan menggunakan bahasa pemrograman python yang telah diimplementasikan.

## 1. Proses Input

Proses input dapat dilakukan dengan input manual ataupun inputan file dari pengguna. Pada bagian ini akan dilakukan pengujian dengan testing inputan file sebagai berikut:

```
3
X Y
R1(X)
W2(X)
W2(Y)
W3(Y)
W1(Y)
C1
C2
C3
```

Baris pertama menyatakan jumlah transaksi, baris kedua menyatakan item, dan baris sisanya menyatakan *schedule* yang akan diperiksa dengan algoritma Lock. Berikut hasil eksekusinya.



```
-----  
Selamat datang pada simulasi Concurrency Control Kelompok 7 K02!  
-----
```

Pilih metode yang ingin digunakan :

1. Simple Locking
2. Serial Optimistic Concurrency Control (OCC)
3. Multiversion Concurrency Control (MVCC)

Nomor Metode yang ingin digunakan: 1

Anda memilih metode Simple Locking

Apakah menggunakan Wound-Wait Scheme? :

1. Ya
2. Tidak

Masukkan Pilihan Anda : 1

Pilih Metode Input :

1. Input Manual
2. Input File

Masukkan Pilihan Anda : 2

Masukkan nama file (Pastikan file sudah ada di folder test) : test3.txt

Schedule 1 : R1(X)

Schedule 2 : W2(X)

Schedule 3 : W2(Y)

Schedule 4 : W3(Y)

Schedule 5 : W1(Y)

Schedule 6 : C1

Schedule 7 : C2

Schedule 8 : C3

-----Simple Locking-----

Berikut Urutan Transaksi

Transaksi 1

Transaksi 2

Transaksi 3

Read X pada T1

=> Give Exclusive Lock X to T1

=> Success

Write X pada T2

=> Transaction 2 is waiting for Exclusive Lock X which is in Transaction 1. W2(X) get into the queue...

Write Y pada T2

=> Transaction 2 is waiting for Exclusive Lock X which is in Transaction 1. W2(Y) get into the queue...

Write Y pada T3

=> Give Exclusive Lock Y to T3

=> Success

Write Y pada T1

=> Transaction 1 is waiting for Exclusive Lock Y which is in Transaction 3. W1(Y) get into the queue...

!!!Do Wound-Wait Scheme!!!

Aborting T=> Exclusive Lock Y on transaction 3 release

Write X pada T2

=> Transaction 2 is waiting for Exclusive Lock X which is in Transaction 1. W2(X) get into the queue...

Write Y pada T2

=> Transaction 2 is waiting for Exclusive Lock X which is in Transaction 1. W2(Y) get into the queue...

Write Y pada T1

=> Give Exclusive Lock Y to T1

=> Success

Commit transaksi T1

=> Success...

=> Exclusive Lock X on transaction 1 release

=> Exclusive Lock Y on transaction 1 release

Write X pada T2

=> Give Exclusive Lock X to T2

=> Success

Write Y pada T2

=> Give Exclusive Lock Y to T2

=> Success

Commit transaksi T2

=> Success...

=> Exclusive Lock X on transaction 2 release

=> Exclusive Lock Y on transaction 2 release

!!!Executing aborted transactions!!!

Write Y pada T3

=> Give Exclusive Lock Y to T3

=> Success

Commit transaksi T3

=> Success...

=> Exclusive Lock Y on transaction 3 release

Berikut Urutan Schedule

XL1(X); R1(X); XL3(Y); W3(Y); A3; UL3(Y); XL1(Y); W1(Y); UL1(X); UL1(Y); C1; XL2(X); W2(X); XL2(Y); W2(Y); UL2(X); UL2(Y); C2;

XL3(Y); W3(Y); UL3(Y); C3;

Validasi Transaksi Sukses Dilakukan

-----Program Selesai-----

## b. Serial Optimistic Concurrency Control (OCC)

*Serial optimistic concurrency control* (OCC) memperbolehkan transaksi untuk mengubah data tanpa memeriksa kemungkinan konflik yang terjadi dengan transaksi lain yang telah melakukan *commit* terlebih dahulu pada saat mengubah data. Konflik hanya akan diperiksa pada saat *commit* transaksi. *Serial optimistic concurrency control* (OCC) bekerja dengan asumsi bahwa banyak transaksi dapat sering berhasil tanpa mengganggu satu sama lain. Jika ditemukan ada konflik pada saat melakukan *commit*, transaksi akan melakukan *rollback* dan *restart*.

Berikut implementasi kode program pada *Serial optimistic concurrency control* (OCC) dengan menggunakan bahasa pemrograman python.

2. **Kelas Transaction** untuk menginisialisasi semua variabel awal yang diperlukan pada saat transaksi, yaitu waktu mulai, waktu selesai, dan validasi.

```
class Transaction(object):

    def __init__(self, id, mulai, selesai, validasi):
        super(Transaction, self).__init__()
        #inisiasi output dan transaksi
        self.id = id
        self.mulai = mulai
        self.selesai = selesai
        self.validasi = validasi
        #read variables held by this transaction
        self.readTransaksi = []
        #write variables held by this transaction
        self.write
```

3. **Proses Validasi OCC** yang diperlukan untuk melakukan validasi terhadap schedule yang ada apakah semua transaksi yang dilakukan sukses atau tidak

```

from Transaction import Transaction
#Library untuk memberikan warna pada program
from colorama import init
init()
from colorama import Fore, Back, Style
import time

def validasiTransaksi(idTransaksi, listTransaksi) -> bool:
    for i in listTransaksi:
        if (i.validasi == None):
            continue
        if (i.id == idTransaksi.id):
            continue
        if(i.validasi < idTransaksi.validasi):
            if(i.selesai < idTransaksi.mulai):
                pass
            elif((idTransaksi.mulai < i.selesai) and (i.selesai <
idTransaksi.validasi)):
                for list in i.writeTransaksi:
                    if list in idTransaksi.readTransaksi:
                        return False
                else:
                    return False
    return True

```

#### 4. Proses OCC

```

def OCCTransaction(schedule, totalTransaksi):

print(Fore.BLUE+"-----

```

```

---"+Fore.RESET)
    print(Fore.RED+"-----Serial Optimistic Concurrency
Control-----"+Fore.RESET)

print(Fore.BLUE+"-----
---"+Fore.RESET)

# Inisialisasi list transaksi
daftarTransaksi = []
for i in range (totalTransaksi):
    transaksi = Transaction(i+1,None,None,None)
    daftarTransaksi.append(transaksi)

# Fase Read
for i in range (len(schedule)):
    if (schedule[i][0] == "C"):
        print(Fore.BLUE+Back.WHITE+"Commit"+Fore.RESET +
Back.RESET+" ", end="")
        print("transaksi T", end="")
        print(schedule[i][1])
        id = int(schedule[i][1]) - 1
        time.sleep(0.1)
        daftarTransaksi[id].validasi = time.time()
        # Fase Validasi
        hasilValidasi = validasiTransaksi(daftarTransaksi[id],
daftarTransaksi)
        # Fase Write
        if (hasilValidasi):
            time.sleep(0.1)
            daftarTransaksi[id].selesai = time.time()
            print(f"==> Transaksi T{id+1} sukses dilakukan")
        else:

```

```

        print(f"==> Transaksi T{id+1} gagal dilakukan")
        print(f"Transaksi T{id+1} aborted")
        return False

    elif (schedule[i][0] == "R" or schedule[i][0] == "W"):
        id = int(schedule[i][1]) - 1
        # Ketika timestamp transaksi belum dimulai
        if (daftarTransaksi[id].mulai == None):
            # Inisiasi waktu mulai transaksi
            time.sleep(0.1)
            daftarTransaksi[id].mulai = time.time()

        if(len(schedule[i]) == 5):
            if (schedule[i][0] == "R"):

                print(Fore.YELLOW+Back.WHITE+"Read"+Fore.RESET +
Back.RESET+" ", end="")
                print(schedule[i][3], "pada transaksi T", end="")
                print(schedule[i][1])

            daftarTransaksi[id].readTransaksi.append(schedule[i][2])
            else:
                print(Fore.RED+Back.WHITE+"Write"+Fore.RESET +
Back.RESET+" ", end="")
                print(schedule[i][3], "pada transaksi T", end="")
                print(schedule[i][1])

            daftarTransaksi[id].writeTransaksi.append(schedule[i][2])
            elif(len(schedule[i]) == 6):
                if (schedule[i][0] == "R"):
                    print(Fore.YELLOW+Back.WHITE+"Read"+Fore.RESET +
Back.RESET+" ", end="")

```

```

        print(schedule[i][4], "pada transaksi T", end="")
        print(schedule[i][1])

daftarTransaksi[id].readTransaksi.append(schedule[i][1])
    else:
        print(Fore.RED+Back.WHITE+"Write"+Fore.RESET +
Back.RESET+" ", end="")
        print(schedule[i][4], "pada transaksi T", end="")
        print(schedule[i][1])

daftarTransaksi[id].writeTransaksi.append(schedule[i][1])
    return True

def formatting(schedule):
    for i in range (len(schedule)):
        kata = schedule[i]
        hurufPertama = kata[0]

        if (hurufPertama != "R"):
            if (hurufPertama != "W"):
                if (hurufPertama != "C"):
                    return False

        if (hurufPertama == "R" or hurufPertama == "W"):
            if (len(kata) == 5) :
                if not(kata[1].isnumeric()):
                    return False
                if kata[2]!="(" or kata[-1]!=")":
                    return False
                if not(kata[-2].isalpha()):
                    return False
            elif (len(kata) == 6):

```

```

        if not(kata[1].isnumeric()):
            return False
        if not(kata[2].isnumeric()) :
            return False
        if kata[3]!="(" or kata[-1]!=")":
            return False
        if not(kata[-2].isalpha()):
            return False
    else:
        return False
else :
    if not(kata[1].isnumeric()):
        return False

    return True
def executeOCC():
print(Fore.BLUE+"-----")
---"+Fore.RESET)
    print(Fore.RED + "----Starting Serial Optimistic Concurrency
Control-----"+Fore.RESET)

print(Fore.BLUE+"-----")
---"+Fore.RESET)
    print(Fore.CYAN+"Masukkan jumlah transaksi : "+Fore.RESET,
end="")
    totalTransaksi = int(input())
    schedule = []
    print(Fore.CYAN+"Masukkan total schedule : "+Fore.RESET, end="")
    totalSchedule = int(input())
print(Fore.BLUE+"-----")
---"+Fore.RESET)
    print(Fore.YELLOW + "Format Schedule : 'R1(X)', 'W1(X)',

```



```

'C1'+Fore.RESET)

print(Fore.BLUE+"-----
---"+Fore.RESET)

    for i in range (totalSchedule ) :
        print("Schedule", (i+1),": ",end="")
        x = str(input())
        schedule.append(x)

    if not(formatting(schedule)):
        print(Fore.RED+"Format yang Anda Masukkan
Salah!!!"+Fore.RESET)

    if (OCCTransaction(schedule, totalTransaksi)):
        print(Fore.BLUE+"Validasi Transaksi Sukses
Dilakukan"+Fore.RESET)
    else :
        print(Fore.RED+"Validasi Transaksi Gagal
Dilakukan"+Fore.RESET)

def pilihMetodeInputOCC():
    inputMetode = 0
    while (inputMetode < 1 or inputMetode > 2):
        print(Fore.BLUE+"Pilih Metode Input : "+Fore.RESET)
        print("1. Input Manual")
        print("2. Input File")
        print(Fore.YELLOW+"Masukkan Pilihan Anda : "+Fore.RESET,
end="")
        inputMetode = int(input())
        if(inputMetode < 1 or inputMetode > 2):
            print(Fore.RED+"")
            print("Input salah! Silahkan masukkan nomor metode yang

```

```

ingin digunakan!"+Fore.RESET)

    if (inputMetode == 1):
        executeOCC()
    elif (inputMetode == 2):
        print("Masukkan nama file (Pastikan file sudah ada di folder
test) : ", end="")
        filename = str(input())
        file = open("../test/" + filename, "r")
        content = file.read()
        arrString = content.split('\n')

        totalTransaksi = int(arrString[0])
        totalSchedule = int(len(arrString)-2)
        schedule = []

        for i in range (totalSchedule):
            x = str(arrString[i+2])
            print("Schedule", (i+1),": ",x)
            schedule.append(x)

        if (OCCTransaction(schedule, totalTransaksi)):
            print(Fore.BLUE+"Validasi Transaksi Sukses
Dilakukan"+Fore.RESET)
        else :
            print(Fore.RED+"Validasi Transaksi Gagal
Dilakukan"+Fore.RESET)

```

## 5. Proses Testing OCC

Proses testing dapat dilakukan dengan input manual ataupun inputan file dari pengguna. Pada bagian ini akan dilakukan pengujian dengan testing inputan file sebagai berikut:

```
3
X Y
R1(X)
W2(X)
W2(Y)
W3(Y)
W1(X)
C1
C2
C3
```

Baris pertama menyatakan jumlah transaksi, baris kedua menyatakan item, dan baris sisanya menyatakan *schedule* yang akan diperiksa dengan algoritma OCC. Berikut hasil eksekusinya.

```
-----
Selamat datang pada simulasi Concurrency Control Kelompok 7 K02!
-----
```

Pilih metode yang ingin digunakan :

1. Simple Locking
2. Serial Optimistic Concurrency Control (OCC)
3. Multiversion Concurrency Control (MVCC)

Nomor Metode yang ingin digunakan: 2

Anda memilih metode Serial Optimistic Concurrency Control (OCC)

Pilih Metode Input :

1. Input Manual
2. Input File

Masukkan Pilihan Anda : 2

Masukkan nama file (Pastikan file sudah ada di folder test) :

test1.txt

Schedule 1 : R1(X)

Schedule 2 : W2(X)

Schedule 3 : W2(Y)

Schedule 4 : W3(Y)

Schedule 5 : W1(X)

Schedule 6 : C1

Schedule 7 : C2

**Schedule 8 : C3**

-----  
-----**Serial Optimistic Concurrency Control**-----  
-----

Read X pada transaksi T1

Write X pada transaksi T2

Write Y pada transaksi T2

Write Y pada transaksi T3

Write X pada transaksi T1

Commit transaksi T1

==> Transaksi T1 sukses dilakukan

Commit transaksi T2

==> Transaksi T2 sukses dilakukan

Commit transaksi T3

==> Transaksi T3 sukses dilakukan

**Validasi Transaksi Sukses Dilakukan**

-----  
-----**Program Selesai**-----  
-----

### 3. Hasil Eksplorasi Recovery

#### a. *Write-Ahead Log*

*Write-ahead log* adalah teknik untuk melakukan *recovery* saat *log* berada pada *main memory* dan belum ditulis ke *stable storage*. Teknik ini dapat mengurangi jumlah *disk write* dan *cost* sinkronisasi. Pada dasarnya, konsep dari teknik ini adalah dengan menuliskan *log* perubahan pada data terlebih dahulu sebelum menuliskan perubahan tersebut. Terdapat tiga aturan yang harus dipenuhi pada *write-ahead log*, *log* yang berkaitan dengan suatu transaksi sudah berada di *stable storage* sebelum *log record commit* transaksi tersebut ditulis ke *stable storage*, transaksi berada pada *commit state* jika dan hanya jika *log record* operasi *commit* transaksi tersebut sudah ditulis ke *stable storage*, serta *log record* data pada suatu blok harus berada di *stable storage* terlebih dahulu sebelum blok data tersebut ditulis ke *database*. Dengan teknik ini, database dapat di-revoker dengan menerapkan perubahan yang tercatat dalam *log*.

PostgreSQL menyimpan *write-ahead log* di dalam direktori `pg_wal/` dalam direktori data *cluster* untuk setiap waktu dan mencatat perubahan yang dilakukan pada database.

#### b. *Continuous Archiving*

*Continous archiving* adalah salah satu metode melakukan *backup* secara berkala. Metode ini bekerja dengan dua bagian, yaitu *base backup* dan *WAL archive*. *Base backup* berisi salinan seluruh *database* pada satu waktu tertentu, sedangkan *WAL archive* terdiri dari semua *WAL file* berisi *log* transaksi yang dilakukan setelah *base backup* telah dibuat.

#### c. *Point-in-Time Recovery*

*Point-in-time recovery* adalah melakukan *recovery* data menjadi kondisi data pada satu waktu tertentu. *Point-in-time recovery* dapat dilakukan dengan adanya *continous archiving*, sehingga dapat melakukan *recovery* pada waktu kapanpun selama pada waktu tersebut *base backup* telah tertulis di *stable storage*. Ketika melakukan *recover*, *point-in-time recovery* bekerja dengan menulis ulang *write-ahead log* yang tertulis sebelumnya secara berurutan hingga waktu yang ditentukan. Untuk menyediakan ruang bagi pengguna melakukan *recovery* ke data sebelum dilakukan *recovery*, terdapat pencatatan *timeline*. Ketika sebuah *recovery* selesai dilakukan,

sebuah *timeline* baru dibuat sehingga kondisi data sebelum *recovery* dan setelah *recovery* dapat dibedakan.

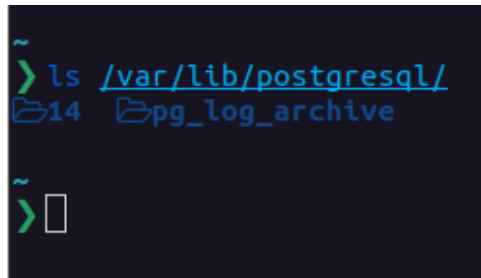
#### d. Simulasi Kegagalan pada PostgreSQL

##### Set-up PostgreSQL untuk mendukung recovery

1. Membuat direktori penyimpanan hasil archive bernama `pg_log_archive`

```
$ sudo -H -u postgres mkdir /var/lib/postgresql/pg_log_archive
```

Setelah itu, terbentuk direktori baru bernama `pg_log_archive` pada direktori `/var/lib/postgresql`



```
~  
> ls /var/lib/postgresql/  
14 pg_log_archive  
  
~  
> █
```

2. Mengubah konfigurasi pada `/etc/postgresql/14/main/postgresql.conf`

```
$ sudo nano /etc/postgresql/14/main/postgresql.conf
```

Konfigurasi dapat diubah dengan *uncomment* bagian yang bersangkutan sebagai berikut

```
wal_level = replica  
  
archive_mode = on  
  
archive_command = 'test ! -f /var/lib/postgresql/pg_log_archive/%f && cp  
%p /var/lib/postgresql/pg_log_archive/%f'
```

```
GNU nano 6.2 /etc/postgresql/14/main/postgresql.conf *
#----->
# WRITE-AHEAD LOG
#----->
# - Settings -
wal_level = replica          # minimal, replica, or logical
                             # (change requires restart)
#fsync = on                  # flush data to disk for crash safety
                             # (turning this off can cause
                             # unrecoverable data corruption)
#synchronous_commit = on    # synchronization level;
                             # off, local, remote_write, remote_ap>
#wal_sync_method = fsync     # the default is the first option
                             # supported by the operating system:
                             #   open datasync
[ line 205/799 (25%), col 1/71 ( 1%), char 7750/29057 (26%) ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify
```

```
GNU nano 6.2 /etc/postgresql/14/main/postgresql.conf *
#checkpoint_warning = 30s    # 0 disables
max_wal_size = 1GB
min_wal_size = 80MB
# - Archiving -
archive_mode = on            # enables archiving; off, on, or always
                             # (change requires restart)
archive_command = 'test ! -f /var/lib/postgresql/pg_log_archive/%f && cp %p >
                             # placeholders: %p = path of file to archive
                             #               %f = file name only
                             # e.g. 'test ! -f /mnt/server/archivedir/%f &
#archive_timeout = 0         # force a logfile segment switch after this
                             # number of seconds; 0 disables
# - Archive Recovery -
[ line 247/799 (30%), col 1/174 ( 0%), char 9345/29149 (32%) ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify
```

### 3. Restart service PostgreSQL

Setelah dilakukan pengubahan konfigurasi, service PostgreSQL perlu dinyalakan untuk mengaplikasikan perubahan konfigurasi.

```
$ sudo systemctl restart postgresql
```

## Backup dan Simulasi kegagalan

### 1. Membuat basis data dan tabel

Menggunakan antarmuka `psql` dengan *user* `postgres`

```
$ sudo -u postgres psql
```

Sebagai contoh, akan dibuat database `sample_db` yang berisi tabel `sample_table`.

```
psql (14.5 (Ubuntu 14.5-0ubuntu0.22.04.1))
Type "help" for help.

postgres=# CREATE DATABASE sample_db;
CREATE DATABASE
postgres=# \c sample_db;
You are now connected to database "sample_db" as user "postgres".
sample_db=# CREATE TABLE sample_table (
sample_db(# id INTEGER,
sample_db(# name CHARACTER VARYING(100)
sample_db(# );
CREATE TABLE
sample_db=# INSERT INTO sample_table(id, name) values
sample_db-# (1, 'item 1'),
sample_db-# (2, 'item 2'),
sample_db-# (3, 'item 3');
INSERT 0 3
sample_db=# SELECT * FROM sample_table;
 id | name
----+-----
  1 | item 1
  2 | item 2
  3 | item 3
(3 rows)

sample_db=#
```

## 2. Mengarsipkan *write-ahead log*

```
$ sudo -u postgres psql -c "select pg_switch_wal();"
```

```
> sudo -u postgres psql -c "select pg_switch_wal();"
could not change directory to "/home/alifia": Permission denied
pg_switch_wal
-----
 0/2B000078
(1 row)
```

## 3. Melakukan backup cluster basis data

```
$ sudo -u postgres pg_basebackup -Ft -D
/var/lib/postgresql/db_file_backup
```

Setelah melakukan backup, akan ada direktori `db_file_backup` pada direktori `/var/lib/postgresql`.



```

~
> ls /var/lib/postgresql/
14 db_file_backup pg_log_archive

~
> sudo ls /var/lib/postgresql/db_file_backup/
backup_manifest base.tar pg_wal.tar

~
>

```

4. Mematikan service PostgreSQL

```
$ sudo systemctl stop postgresql
```

5. Menghapus data pada direktori kluster data PostgreSQL

```
$ sudo rm /var/lib/postgresql/14/main -r
```

Perintah ini akan mengosongkan seluruh data di direktori kluster data  
/var/lib/postgresql/14/main

```

~
> sudo ls /var/lib/postgresql/14/main

~
>

```

## Melakukan recovery

1. Pada file *backup* yang telah dibuat, *unzip* file *base.tar* di  
/var/lib/postgresql/14/main dan file *pg\_wal.tar* di  
/var/lib/postgresql/14/main/pg\_wal/

```

$ sudo tar xvf /var/lib/postgresql/db_file_backup/base.tar -C
/var/lib/postgresql/14/main/

$ sudo tar xvf /var/lib/postgresql/db_file_backup/pg_wal.tar -C
/var/lib/postgresql/14/main/pg_wal/

```

```

~
> sudo ls /var/lib/postgresql/14/main/
backup_label  pg_multixact  pg_stat_tmp  pg_xact
base          pg_notify     pg_subtrans  postgresql.auto.conf
global        pg_replslot   pg_tblspc    tablespace_map
pg_commit_ts  pg_serial     pg_twophase
pg_dynshmem   pg_snapshots  PG_VERSION
pg_logical    pg_stat       pg_wal

~
> sudo ls /var/lib/postgresql/14/main/pg_wal
000000010000000000000002A archive_status

~
> █

```

2. Buat file `recovery.conf` di folder `/var/lib/postgresql/10/main`

```
$ sudo nano /var/lib/postgresql/14/main/recovery.conf
```

Isi file tersebut dengan

```
restore_command = 'cp /var/lib/postgresql/pg_log_archive/%f %p'
```

```

~
> sudo cat /var/lib/postgresql/14/main/recovery.conf
restore_command = 'cp /var/lib/postgresql/pg_log_archive/%f %p'

```

3. Menyalakan kembali *service* PostgreSQL

```
$ sudo systemctl start postgresql
```

4. Cek hasil recovery basis data

```

psql (14.5 (Ubuntu 14.5-0ubuntu0.22.04.1))
Type "help" for help.

postgres=# \l
                                List of databases
  Name      | Owner   | Encoding | Collate | Ctype   | Access privileges
-----+-----+-----+-----+-----+-----
postgres   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | 
sample_db  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | 
template0  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres      +
template1  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | postgres=CTc/postgres +
(4 rows)

postgres=# \c sample_db;
You are now connected to database "sample_db" as user "postgres".
sample_db=# \d
                                List of relations
 Schema | Name      | Type  | Owner
-----+-----+-----+-----
 public | sample_table | table | postgres
(1 row)

sample_db=# select * from sample_table;
 id | name
----+-----
  1 | item 1
  2 | item 2
  3 | item 3
(3 rows)

sample_db=# 

```

## 4. Kesimpulan dan Saran

### a. Kesimpulan

Berdasarkan hasil eksplorasi Concurrency Control dan Recovery pada PostgreSQL serta implementasi Concurrency Control Protocol dalam tugas besar ini, dapat diambil kesimpulan yang dijabarkan dalam poin-poin berikut:

1. Jenis isolasi transaksi pada PostgreSQL terdiri dari Serializable, Repeatable Read, dan Read Committed. PostgreSQL tidak mengimplementasikan jenis isolasi Read Uncommitted dan pilihan bawaan dalam membuat transaksi adalah jenis Read Committed
2. Implementasi *Simple Locking* (SL) dilakukan dengan mengimplementasikan lock pada tiap data dan pada tiap pengaksesan datanya dilakukan pengecekan validitas akses data dengan melihat apakah data tersebut telah dilock transaksi lain atau belum.
3. Implementasi *Simple Locking* (SL) terdapat kemungkinan terjadi *deadlock*. Jika ada 2 transaksi yang saling menunggu data untuk dilepas locknya, misal transaksi a menunggu transaksi b untuk melepas *lock* pada data X dan transaksi b menunggu transaksi a untuk melepas *lock* pada data Y, akan terjadi *deadlock* karena kedua transaksi tersebut saling menunggu transaksi lain untuk melepas *lock* pada data yang diinginkan.
4. Implementasi *Serial optimistic concurrency control* (OCC) dilakukan dengan menambahkan *timestamp* pada tiap transaksi dan mengecek validitas akses data tiap transaksi terhadap semua transaksi pada *schedule*.
5. Recovery dilakukan untuk mengembalikan *state*/keadaan data pada suatu waktu yang biasanya dilakukan karena data tersebut hilang/rusak.
6. Pada PostgreSQL, terdapat tiga metode recovery yaitu *write-ahead log* yang menuliskan *log* sebelum melakukan perubahan data, *continuous archiving* yang melakukan *backup* secara berkala, serta *point-in-time recovery* yang melakukan *recovery* basis data menjadi kondisi basis data tersebut pada waktu tertentu.

## **b. Saran**

Saran yang dapat kami berikan setelah melaksanakan tugas ini adalah sebaiknya tugas yang diberikan lebih diketahui lagi behaviour-nya seperti apa dan perlu dilakukan optimasi pada implementasi algoritma OCC dan Simple Locking.

## 5. Pembagian Kerja

Berikut adalah pembagian tugas pada kelompok kami.

<b>Nama</b>	<b>NIM</b>	<b>Pembagian Tugas</b>
Fadil Fauzani	13520032	1. Eksplorasi Concurrency Control 2. Kesimpulan
Shadiq Harwiz	13520038	1. <i>Simple Locking (exclusive only)</i> 2. Saran
Alifia Rahmah	13520122	Eksplorasi <i>recovery</i>
Febryola Kurnia Putri	13520140	1. <i>Serial Optimistic Concurency Control (OCC)</i> 2. Petunjuk penggunaan program

## Referensi

Slide Mata Kuliah IF3140 Manajemen Basis Data Tahun 2022/2023

<https://www.postgresql.org/docs/current/wal-intro.html> diakses tanggal 4 Desember 2022

<https://www.postgresql.org/docs/9.1/continuous-archiving.html> diakses tanggal 25 November 2022

<https://malisper.me/postgres-backups-with-continuous-archiving/> diakses tanggal 25 November 2022

<https://www.scalingpostgres.com/tutorials/postgresql-backup-point-in-time-recovery/> diakses tanggal 25 November 2022

<https://www.postgresql.org/docs/current/transaction-iso.html> diakses tanggal 4 November 2022

<https://www.postgresql.org/docs/current/sql-set-transaction.html> diakses tanggal 4 November 2022