

# **Mekanisme *Concurrency Control* dan *Recovery***

Tugas Besar 2 IF3140 Manajemen Basis Data



**Kelas 3 - Kelompok 14**

Aji Andhika Falah	13520012
Louis Yanggara	13520063
Adelline Kania Setiyawan	13520084
Gerald Abraham Sianturi	13520138

**Program Studi Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung**

**Jl. Ganesha 10, Bandung 40132**

2022

## DAFTAR ISI

### Bab 1

#### Eksplorasi dan implementasi concurrency protocol serta eksplorasi recovery

	3
1.1 Eksplorasi concurrency control	3
1.1.1 Serializability	3
1.1.2 Repeatable Read	4
1.1.2 Read Committed	4
1.1.2 Read Uncommitted	5
1.2 Implementasi Concurrency Control	8
1.2.1 Simple locking	8
1.2.1.1 Deskripsi protokol	8
1.2.1.2 Implementasi	9
1.2.1.3 Hasil percobaan	18
1.2.2 Serial optimistic concurrency protocol	20
1.2.2.1 Deskripsi protokol	20
1.2.2.2 Implementasi	21
1.2.2.3 Hasil percobaan	25
1.3 Eksplorasi Recovery	27
1.3.1 Write-Ahead Log	27
1.3.2 Continuous Archiving	28
1.3.3 Point in Time Recovery	28
1.3.4 Simulasi Kegagalan	29

### Bab 2

#### Kesimpulan dan Saran

2.1 Kesimpulan	37
2.2 Saran	37

#### Pembagian Kerja

38

#### Referensi

39

# Bab 1

## Eksplorasi dan implementasi *concurrency protocol* serta eksplorasi *recovery*

### 1.1 Eksplorasi *concurrency control*

#### 1.1.1 Serializability

Derajat isolasi *Serializability* merupakan derajat isolasi yang paling ketat. *Serializability* berusaha untuk menirukan eksekusi transaksi serial untuk semua transaksi yang ada seolah-olah semua transaksi dieksekusi satu per satu secara serial bukan secara konkuren. Namun, seperti halnya pada derajat isolasi *Repeatable Read*, aplikasi yang menggunakan derajat isolasi *Serializability* harus memiliki penanganan dan persiapan untuk mengulangi transaksi jika terjadi kegagalan. Sebenarnya, *Serializability* bekerja sama persis dengan *Repeatable Read* kecuali *Serializability* memantau kondisi yang dapat menyebabkan eksekusi transaksi serial secara konkuren yang tidak konsisten dengan semua kemungkinan eksekusi serial pada transaksi. Pemantauan ini tidak menyebabkan adanya pemblokiran dari yang ada pada *Repeatable Read*, namun ada tambahan (*overhead*) pada pemantauan dan deteksi terhadap kondisi yang dapat menyebabkan adanya anomali pada *Serialization* yang dapat menyebabkan kegagalan.

Ketika *Serializability* digunakan untuk mencegah anomali, sangat penting bahwa data yang dibaca dari tabel tidak dianggap valid hingga transaksi berhasil di-*commit*. Untuk menjamin *Serializability*, PostgreSQL menggunakan *predicate locking* yang berarti mempertahankan *lock* yang memberi izin untuk menentukan apakah sebuah *write* dapat memberi dampak pada hasil *read* sebelumnya pada sebuah transaksi konkuren yang berjalan duluan. Pada PostgreSQL, *predicate lock* tidak menyebabkan *blocking* sehingga tidak dapat menyebabkan *deadlock*.

Derajat isolasi *Serializability* diimplementasikan dengan menggunakan teknik yang dikenal dengan *Serializable Snapshot Isolation* yang dibangun di atas *Snapshot Isolation* dengan menambahkan anomali *serializability*.

### 1.1.2 Repeatable Read

Derajat isolasi *Repeatable Read* hanya melihat data yang di-*commit* sebelum transaksi dimulai, ia tidak pernah melihat data yang belum di-*commit* maupun perubahan yang di-*commit* saat transaksi dieksekusi oleh transaksi yang konkuren. Namun *Repeatable Read* tetap melihat hasil eksekusi dari transaksinya sendiri meskipun belum di-*commit*. Hal ini lebih memenuhi standar dari SQL dan mencegah terjadinya fenomena *Dirty Read*, *Non Repeatable Read*, *Phantom Read* namun tidak mencegah terjadinya anomali *serialization*.

*Repeatable Read* menjamin bahwa setiap transaksi melihat *view* basis data yang stabil, namun *view* tidak harus selalu konsisten dengan setiap eksekusi serial pada transaksi konkuren pada level yang sama. Sebagai contoh, sebuah transaksi *read-only* pada level ini dapat melihat kontrol dari *record* yang sudah diperbaharui untuk menunjukkan bahwa *batch* sudah selesai dijalankan namun tidak melihat detail dari *record* yang secara logika merupakan bagian dari *batch* karena ia membaca versi awal dari *control record*.

*Repeatable Read* diimplementasikan dengan menggunakan teknik yang bernama *Snapshot Isolation*. Perbedaan pada tingkah laku dan performa dapat diobservasi ketika dibandingkan dengan sistem yang menggunakan teknik *locking* tradisional yang mengurangi konkuren.

### 1.1.2 Read Committed

*Read Committed* merupakan derajat isolasi yang digunakan oleh PostgreSQL secara *default*. Ketika sebuah transaksi menggunakan *Read Committed*, *query* SELECT (tanpa FOR UPDATE/SHARE *clause*) akan melihat data yang sudah di-*commit* sebelum *query* dimulai. Transaksi tidak dapat melihat data yang belum di-*commit* maupun perubahan *commit* selama *query* dieksekusi oleh transaksi konkuren. Akibatnya, *query* SELECT melihat *snapshot* basis data tepat saat *query* mulai berjalan. Namun, *query* SELECT tetap melihat efek dari perubahan yang dieksekusi oleh transaksinya meskipun belum di-*commit*. Selain itu, dua SELECT yang berhasil dapat melihat data yang berbeda meskipun keduanya berada pada transaksi yang sama. Jika transaksi lain melakukan *commit* terhadap perubahan setelah SELECT yang pertama dan sebelum SELECT yang kedua.

Karena derajat isolasi *Read Committed* memulai setiap *command* dengan *snapshot* yang baru termasuk semua transaksi yang sudah di-*commit* pada saat itu, *command* selanjutnya pada

transaksi yang sama akan melihat efek dari transaksi konkuren yang sudah di-*commit*. Permasalahan utamanya adalah apakah sebuah *command* melihat basis data yang konsisten.

### 1.1.2 Read Uncommitted

*Read Uncommitted* merupakan derajat isolasi paling lemah pada PostgreSQL yang dapat menyebabkan semua fenomena yang ada dan yang seharusnya dihindari pada basis data. Jika pada *Read Committed locking* diimplementasikan dengan mengambil *shared locks* jangka pendek ketika membaca data, pada *Read Uncommitted* tidak ada *shared locks* yang digunakan sama sekali sehingga dapat menyebabkan *dirty read*.

Berikut ini adalah tabel perbandingan setiap derajat isolasi terhadap kemungkinan terjadinya fenomena tertentu:

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read uncommitted	Allowed, but not in PG	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible	Possible
Repeatable read	Not possible	Not possible	Allowed, but not in PG	Possible
Serializable	Not possible	Not possible	Not possible	Not possible

## Simulasi

### Perbandingan *Read Committed* dan *Repeatable Read*

Isolation Level	Hasil
<i>Read Committed</i>	<pre>tubea2=# begin; BEGIN tubea2=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED; SET tubea2=# select * from orders;  id   description   total   status   created_at -----+-----+-----+-----+-----  2   done_order   50   NEW   2016-08-21 07:00:25  3   failed_order   200   FAILED   2016-10-03 12:13:21  1   n   999   N   2016-06-22 19:10:25 (3 rows)  tubea2=# select * from orders;  id   description   total   status   created_at -----+-----+-----+-----+-----  2   done_order   50   NEW   2016-08-21 07:00:25  3   failed_order   200   FAILED   2016-10-03 12:13:21  1   n   999   N   2016-06-22 19:10:25 (3 rows)  tubea2=# select * from orders;  id   description   total   status   created_at -----+-----+-----+-----+-----  3   failed_order   200   FAILED   2016-10-03 12:13:21  1   n   999   N   2016-06-22 19:10:25  2   done_order   500   NEW   2016-08-21 07:00:25 (3 rows)</pre> <pre>tubea2=# begin; BEGIN tubea2=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED; SET tubea2=# UPDATE orders SET total = 500 WHERE id=2; UPDATE 1 tubea2=# commit tubea2=# ; COMMIT tubea2=#</pre>
<i>Repeatable Read</i>	<pre>tubea2=# begin; BEGIN tubea2=# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; SET tubea2=# select * from orders;  id   description   total   status   created_at -----+-----+-----+-----+-----  3   failed_order   200   FAILED   2016-10-03 12:13:21  1   n   999   N   2016-06-22 19:10:25  2   done_order   500   NEW   2016-08-21 07:00:25 (3 rows)  tubea2=# select * from orders;  id   description   total   status   created_at -----+-----+-----+-----+-----  3   failed_order   200   FAILED   2016-10-03 12:13:21  1   n   999   N   2016-06-22 19:10:25  2   done_order   500   NEW   2016-08-21 07:00:25 (3 rows)  tubea2=# commit; COMMIT</pre> <pre>postgres=# \c tubea2; You are now connected to database "tubea2" as user "postgres". tubea2=# begin; BEGIN tubea2=# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;  SET tubea2=# UPDATE orders SET total = 2 WHERE id=3; UPDATE 1 tubea2=# commit; COMMIT tubea2=#</pre>

Untuk mensimulasikan perbedaan antara *Read Committed* dan *Repeatable Read*, digunakan fenomena *NonRepeatable Read* dimana pada *Read Committed* dapat terjadi, namun pada *Repeatable Read* tidak dapat terjadi. Dapat dilihat bahwa pada *Read Committed*, transaksi pada sebelah kiri dapat melihat UPDATE yang dilakukan oleh transaksi kanan setelah di-commit. Sedangkan untuk *Repeatable Read*, transaksi di sebelah kiri tidak dapat melihat UPDATE yang dilakukan meskipun sudah di-commit.

## Perbandingan *Repeatable Read* dan *Serializability*

Isolation Level	Hasil
<i>Repeatable Read</i>	<div> <pre> tubes2=# begin; BEGIN tubes2=# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; SET tubes2=# select * from serial; class   value ----- 1   10 1   20 2   200 2   100 (4 rows)  tubes2=# INSERT INTO serial(class, value) VALUES(2, 30); INSERT 0 1 tubes2=# select * from serial; class   value ----- 1   10 1   20 2   200 2   100 2   30 (5 rows)  tubes2=# commit; COMMIT tubes2=# </pre> </div> <div> <pre> tubes2=# begin; BEGIN tubes2=# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; SET tubes2=# select * from serial; class   value ----- 1   10 1   20 2   200 2   100 (4 rows)  tubes2=# INSERT INTO serial(class, value) VALUES(1, 300); INSERT 0 1 tubes2=# select * from serial; class   value ----- 1   10 1   20 2   200 2   100 1   300 (5 rows)  tubes2=# commit; COMMIT tubes2=# </pre> </div>
<i>Serializability</i>	<div> <pre> tubes2=# begin; BEGIN tubes2=# SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; SET tubes2=# SELECT * FROM serial; class   value ----- 1   10 1   20 2   200 2   100 (4 rows)  tubes2=# INSERT INTO serial(class, value) VALUES(2, 30); INSERT 0 1 tubes2=# SELECT * FROM serial; class   value ----- 1   10 1   20 2   200 2   100 2   30 (5 rows)  tubes2=# COMMIT; COMMIT tubes2=# tubes2=# tubes2=# tubes2=# tubes2=# </pre> </div> <div> <pre> tubes2=# begin; BEGIN tubes2=# SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; SET tubes2=# SELECT * FROM SERIAL; class   value ----- 1   10 1   20 2   200 2   100 (4 rows)  tubes2=# INSERT INTO serial(class, value) VALUES(1, 300); INSERT 0 1 tubes2=# SELECT * FROM serial; class   value ----- 1   10 1   20 2   200 2   100 1   300 (5 rows)  tubes2=# 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. tubes2=# </pre> </div>

Untuk mensimulasikan perbedaan antara *Repeatable Read* dan *Serializability*, digunakan fenomena *Serialization Anomaly* dimana pada *Repeatable Read* dapat terjadi, namun pada *Serializability* tidak dapat terjadi. Dapat dilihat bahwa pada *Repeatable Read*, transaksi pada sebelah kiri dan kanan dapat di-commit meskipun kedua transaksi saling dependen sehingga

menyebabkan *Serialization anomaly*. Sedangkan untuk *Serializability*, hanya salah satu transaksi yang dapat di-*commit* dan transaksi lainnya akan memberi pesan *error* ketika di-*commit*.

## 1.2 Implementasi *Concurrency Control*

### 1.2.1 *Simple locking*

#### 1.2.1.1 Deskripsi protokol

*Simple Locking* adalah sebuah prosedur yang digunakan untuk mengendalikan akses bersamaan ke data. Ketika sebuah transaksi sedang mengakses *database*, sebuah *lock* mungkin menolak akses ke transaksi lain untuk mencegah hasil yang salah. Ada dua macam *lock*, yaitu *shared lock* dan *exclusive lock* yang harus digunakan sebelum melakukan akses membaca ataupun menulis terhadap *database*. Akan tetapi, untuk *simple locking* ini hanya menggunakan *exclusive lock*. Penggunaan *lock* ini adalah untuk menjaga konsistensi data didalam *database*. Jika sebuah transaksi mempunyai sebuah *exclusive lock* pada sebuah data, transaksi tersebut dapat membaca dan mengubah item data.

Cara Kerja dari *simple locking* adalah sebagai berikut:

- Transaksi apapun yang membutuhkan akses pada sebuah item data harus melakukan lock terhadap item tersebut, sebuah *exclusive lock* untuk akses membaca dan menulis.
- Jika item belum dikunci oleh transaksi lain, lock tersebut akan dikabulkan
- Jika item sedang dikunci, DBMS menentukan apakah permintaan ini compatible dengan lock saat ini. Karena pada implementasi ini hanya mengabulkan *exclusive lock*, transaksi harus menunggu sampai lock yang ada terlepas.
- Sebuah transaksi lanjut memegang lock sampai transaksi tersebut melepasnya baik pada waktu eksekusi ataupun pada waktu transaksi tersebut berakhir (abort atau commit). Efek operasi tulis akan terlihat pada transaksi lain hanya pada waktu *exclusive lock* telah dilepas.



### 1.2.1.2 Implementasi

Implementasi dilakukan dengan menggunakan input *file* dari pengguna. Dideklarasikan beberapa variabel yang akan dipakai, yakni `arrOperation` untuk menyimpan semua data yang terlibat pada transaksi, dan `lockManager` untuk mengetahui apakah ada deadlock dan cara mengurusnya. Setiap operation pada `arrOperation` akan dilihat apakah transaksi tersebut memiliki *lock* atau tidak. Jika tidak, maka akan waiting. Jika Iya, akan dijalankan. Transaksi yang waiting akan dijalankan setelah *lock* yang dibutuhkan sudah dilepaskan. Jika ada *deadlock* maka akan di *abort*.

Berikut adalah hasil implementasi kodenya:

```
class Operation:

    def __init__(self, transaction=None, action=None, data=None,
operation=None):
        if (operation):
            self.transaction = operation.transaction
            self.action = operation.action
            self.data = operation.data
        else:
            self.transaction = transaction
            self.action = action
            self.data = data

    def __str__(self):
        data = f'({self.data})' if (self.action == 'R' or
self.action == 'W') else ""
        return f'{self.action}{self.transaction.id}{data}'

class Transaction:

    def __init__(self, id):
        self.id = id
```

```

def __str__(self):
    return 'T' + str(self.id)

def __eq__(self, transaction):
    return (self.id == transaction.id)

# Membaca File
def generalSetup(fileName):
    file = open("./fileInput/" + fileName, "r")
    buff = file.read()
    arrString = buff.split('\n')
    arrTransaction = []
    num_of_transaction = int(arrString.pop(0))
    for i in range(num_of_transaction):
        arrTransaction.append(Transaction(i+1))
    raw_data = arrString.pop(0).split(' ')
    arrOperation = []
    for s in arrString:
        s = s.replace('(', ' ')
        s = s.replace(')', ' ')
        if len(s) > 2:
            arrOperation.append(
                Operation(
                    arrTransaction[int(s[1])-1],
                    s[0],
                    s[2]
                )
            )
        else:
            arrOperation.append(

```

```

        Operation(
            arrTransaction[int(s[1])-1],
            s[0]
        )
    )

    return arrTransaction, arrOperation, raw_data

def SLock_Converter(arrTransaction, arrData, arrString):
    SL_DataContainer = []
    arrDataLabel = []
    for data in arrData:
        data_label = data.data
        arrDataLabel.append(data_label)
        SL_DataContainer.append(SL.SLData(SL.Data(data_label)))
    LockManager = SL.LockManager(SL_DataContainer)
    arrSLTransaction = []
    for transaction in arrTransaction:
        arrSLTransaction.append(SL.SLTransaction(transaction,
LockManager))
    arrOperation = []
    for proc in arrData:
        transaction_id = proc.transaction.id
        action = proc.action
        dataLabel = proc.data

    arrOperation.append(SL.Operation(arrSLTransaction[transaction_id -
1], action, SL_DataContainer[arrDataLabel.index(dataLabel)],
LockManager))

    return arrOperation, LockManager

class Operation:

```

```

def __init__(self, SLTransaction, action, SLData, lockManager):
    self.SLTransaction = SLTransaction
    self.action = action
    self.SLData = SLData
    self.lockManager = lockManager

def run(self):
    if (self.SLTransaction.transaction.id in
self.lockManager.deadlocked_transactions):
        self.lockManager.deadlocked_operation.append(self)
    else:
        success = True
        if (self.action == 'R'):
            success = self.SLTransaction.read(self.SLData)
        elif (self.action == 'W'):
            success = self.SLTransaction.write(self.SLData)
        else:
            success = self.SLTransaction.commit()
        if (not success):
            self.lockManager.pending.append(self)
            if (not isinstance(self.SLData, str) and
len(self.SLData.lock) > 0):
                wait_id = self.SLTransaction.transaction.id
                waitee_id = self.SLData.lock[0]
                deadlock =
self.lockManager.search_deadlock(wait_id, waitee_id)
                if not deadlock:
self.lockManager.deadlock_detector[waitee_id].append(wait_id)
            else:
                if (wait_id > waitee_id):
                    del_id = wait_id

```

```

        else:
            del_id = waitee_id
            print(f'\n!!! ABORTING T{del_id} BECAUSE A
DEADLOCK HAS BEEN DETECTED!!!\n'.format(del_id))
            self.lockManager.delete_lock(del_id)

self.lockManager.deadlocked_transactions.append(del_id)
            length = len(self.lockManager.pending)
            for i in range(length):
                operation =
self.lockManager.pending.pop(0)
                operation.run()

class Transaction:

    def __init__(self, id):
        self.id = id

class Data:

    def __init__(self, label):
        self.label = label

class LockManager:

    def __init__(self, all_data):
        self.all_data = all_data
        self.pending = []
        self.deadlock_detector = {}
        self.deadlocked_transactions = []
        self.deadlocked_operation = []

    def exclusive_lock(self, transaction, data):

```

```

        if (transaction.transaction.id not in
self.deadlock_detector):
            self.deadlock_detector[transaction.transaction.id] = []
            if ((len(data.lock) > 0) and (transaction.transaction.id ==
data.lock[0])):
                return True
            success = True
            for operation in self.pending:
                if (operation.SLTransaction.transaction.id ==
transaction.transaction.id):
                    success = False
                    break
            if (success):
                data.lock.append(transaction.transaction.id)
                if (transaction.transaction.id != data.lock[0]):
self.deadlock_detector[data.lock[0]].append(transaction.transaction
.id)
                    success = False
            return success

def search_deadlock(self, wait_id, wait_id2):
    wait1 = False
    wait2 = False
    if wait_id2 in self.deadlock_detector:
        if wait_id in self.deadlock_detector[wait_id2]:
            wait1 = True
    if wait_id in self.deadlock_detector:
        if wait_id2 in self.deadlock_detector[wait_id]:
            wait2 = True

    return wait1 and wait2

```

```

def delete_lock(self, transaction_id):
    for data in self.all_data:
        if transaction_id in data.lock:
            data.lock.remove(transaction_id)
    self.deadlock_detector.pop(transaction_id, None)

class SLTransaction:

    def __init__(self, transaction, lockManager):
        self.transaction = transaction
        self.lockManager = lockManager

    def write(self, SLData):
        success = self.lockManager.exclusive_lock(self, SLData)
        if (success):
            print(f'W{self.transaction.id}({SLData.data.label})')
            return True
        else:
            print(f'W{self.transaction.id}({SLData.data.label}) is
waiting')
            return False

    def read(self, SLData):
        success = self.lockManager.exclusive_lock(self, SLData)
        if (success):
            print(f'R{self.transaction.id}({SLData.data.label})')
            return True
        else:
            print(f'R{self.transaction.id}({SLData.data.label}) is
waiting')
            return False

```

```

def commit(self):
    success = True
    for operation in self.lockManager.pending:
        if (operation.SLTransaction.transaction.id ==
self.transaction.id):
            success = False
            break

    if (success):
        for index, data in
enumerate(self.lockManager.all_data):
            if (len(data.lock) > 0 and data.granted_lock() ==
self.transaction.id):
                data_pop = data.lock.pop(0)

self.lockManager.deadlock_detector.pop(self.transaction.id, None)
        print(f'C{self.transaction.id}')

        length = len(self.lockManager.pending)
        for i in range(length):
            operation = self.lockManager.pending.pop(0)
            operation.run()
    else:
        print(f'C{self.transaction.id} is waiting')
    return success

class SLData:

    def __init__(self, data):
        self.data = data
        self.lock = []

    def granted_lock(self):

```



```

        return self.lock[0]

def run_SL(filename):
    print('Menghitung Metode Simple Locking')
    T, data, operation_string = Util.generalSetup(filename)
    arrOperation, lockManager = Util.SLock_Converter(T, data,
operation_string)

    for operation in arrOperation:
        operation.run()

    lockManager.deadlocked_transactions = []

    newArrOperation =
copy.deepcopy(lockManager.deadlocked_operation)
    lockManager.deadlocked_operation = []

    while (len(newArrOperation) > 0):
        print('Retry Aborted Transactions')
        for operation in newArrOperation:
            operation.run()
        lockManager.deadlocked_transactions = []
        newArrOperation =
copy.deepcopy(lockManager.deadlocked_operation)
        lockManager.deadlocked_operation = []

    if len(lockManager.deadlocked_transactions) > 0:
        print('\nTransactions:')
        for deadlock in lockManager.deadlocked_transactions:
            print(f'T{deadlock}'.format(deadlock))

```

### 1.2.1.3 Hasil percobaan

1. Kasus 1, dimana tidak terdapat konflik pada *schedule*

*Input:*

```
SimpleLock > fileInput > ≡ dummytx.txt
1      3
2      X Y
3      R1(X)
4      W2(X)
5      W2(Y)
6      W3(Y)
7      W1(X)
8      C1()
9      C2()
10     C3()|
```

*Output:*

```
PS C:\Users\Aji\Documents\GitHub\MBD02\SimpleLock> py SLMain.py
Ketik nama file yang berada pada folder test: dummytx.txt
Menghitung Metode Simple Locking
R1(X)
W2(X) is waiting
W2(Y) is waiting
W3(Y)
W1(X)
C1
W2(X)
W2(Y) is waiting
C2 is waiting
C3
W2(Y)
C2
Metode Simple Locking Selesai
```

2. Kasus 2, terdapat konflik pada *schedule*

*Input:*

```
SimpleLock > fileInput > ≡ dummytx2.txt
1      2
2      E F G
3      R1(E)
4      R2(G)
5      R1(F)
6      R2(F)
7      R1(G)
8      R2(E)
9      W1(F)
10     W2(E)
11     W1(G)
12     C2()
13     C1()
```

*Output:*

```

PS C:\Users\Aji\Documents\GitHub\MBD02\SimpleLock> py SLMain.py
Ketik nama file yang berada pada folder test: dummytx2.txt
Menghitung Metode Simple Locking
R1(E)
R2(G)
R1(F)
R2(F) is waiting
R1(G) is waiting

!!! ABORTING T2 BECAUSE A DEADLOCK HAS BEEN DETECTED!!!

R1(G)
W1(F)
W1(G)
C1
Retry Aborted Transactions
R2(F)
R2(E)
W2(E)
C2
Metode Simple Locking Selesai

```

3. Kasus 3, dimana tidak terdapat konflik pada *schedule*

*Input:*

```

SimpleLock > fileInput > dummytx3.txt
1      2
2      A B
3      R1(B)
4      R2(B)
5      R2(A)
6      R1(A)
7      W1(A)
8      C1
9      W2(B)
10     W2(A)
11     C2

```

*Output:*

```

PS C:\Users\Aji\Documents\GitHub\MBD02\SimpleLock> py SLMain.py
Ketik nama file yang berada pada folder test: dummytx3.txt
Menghitung Metode Simple Locking
R1(B)
R2(B) is waiting
R2(A) is waiting
R1(A)
W1(A)
C1
R2(B)
R2(A)
W2(B)
W2(A)
C2
Metode Simple Locking Selesai

```

### 1.2.2 *Serial optimistic concurrency protocol*

#### 1.2.2.1 Deskripsi protokol

*Serial optimistic concurrency protocol* atau disebut juga *validation based protocol* adalah salah satu teknik pada *concurrency control* untuk melakukan manajemen eksekusi dari operasi-operasi yang terjadi pada *database*. *Serial optimistic concurrency protocol* didesain dengan asumsi transaksi-transaksi yang terjadi pada *database* sangat jarang mengalami konflik atau dengan kata lain suatu transaksi tidak memakan waktu lama (transaksi tidak terlalu kompleks). Teknik *concurrency control* ini merupakan pendekatan yang ideal untuk kasus dimana tidak terdapat konflik pada transaksi-transaksi yang ada dan dapat mengurangi *overhead* dari mengaplikasikan *lock* pada suatu transaksi.

Untuk memahami *serial optimistic concurrency control*, pertama-tama perlu dipahami dua terminologi berikut ini, yakni *write set* dan *read set*. *Write set* adalah himpunan semua operasi *write* pada suatu transaksi T dan *read set* adalah himpunan semua operasi *read* pada transaksi T.

Ide utama dari teknik ini adalah DBMS membuat suatu *local workspace* untuk tiap transaksi dimana setiap objek (data) yang dibaca akan di-copy ke *workspace* ini atau jika objek dimodifikasi, maka modifikasi berlaku hanya pada *local workspace* (bukan pada *global database*). Lalu, ketika transaksi di-commit, DBMS akan membandingkan apakah pada *write set* yang ada pada *local workspace* terdapat konflik dengan transaksi lainnya. Jika tidak terdapat konflik, *write set* akan diaplikasikan ke *database* secara global.

Fase-fase yang ada pada teknik ini meliputi *read phase*, yakni nilai-nilai tertentu di-copy ke *local workspace* atau nilai pada *workspace* diubah. Lalu, fase kedua adalah *validation phase* dimana terjadi ketika suatu transaksi di-commit dan akan dilakukan pengecekan apakah terdapat konflik dengan transaksi lain. Lalu, fase terakhir dilakukan jika validasi sukses dan dilakukan *push* ke *global database* dari *local workspace*. Algoritma yang digunakan untuk mengimplementasikan teknik ini berpangku pada data *timestamp* pada tiap transaksi, di antaranya,

1. START: menunjukkan waktu pertama kali operasi (*read* atau *write*) dilakukan pada suatu *local workspace* dari transaksi tersebut
2. VALIDATION/TS: menunjukkan waktu sesaat setelah *commit* yang menunjukkan berakhirnya *read phase*
3. FINISH: menunjukkan waktu setelah perubahan (*write*) pada *global database* dari *local workspace*

#### 1.2.2.2 Implementasi

Implementasi dilakukan dengan menggunakan input *file* dari pengguna. Dideklarasikan beberapa variabel yang akan dipakai, yakni `numTxn` yang berguna untuk melakukan pengecekan apakah semua transaksi sudah tervalidasi, `dataTxn` untuk menyimpan semua data yang terlibat pada transaksi, `txnSequence` untuk mengetahui urutan transaksi, `validatedTxnId` untuk menyimpan transaksi yang telah divalidasi yang akan digunakan bersama `numTxn` untuk *exit* program kasus sukses pada semua transaksi, lalu `txn` yang berisi *list* dari tiap transaksi yang masing-masing elemennya merupakan suatu Class `Transaction` yang memiliki properti `read_set`, `write_set`, dan properti lain untuk menyimpan *timestamp* (START, VALIDATION atau TS, dan FINISH) dari transaksi.

Algoritma validasi pada suatu transaksi  $T_i$  dilakukan dengan mengecek semua transaksi  $T_j$  yang nilai *timestamp* dari VALIDATION atau TS dari  $T_j$  lebih kecil dari  $T_i$  (atau dengan kata lain semua transaksi  $T_j$  dimana *validation* dilakukan sebelum  $T_i$ . Pengecekan validasi berhasil didasarkan pada pemenuhan salah satu kondisi di bawah ini,

1.  $FINISH(T_j) < START(T_i)$
2.  $FINISH(T_j) < VALIDATION(T_i)$  dan *read set* dari  $T_i$  *disjoint* dengan *write set*  $T_j$  (tidak ada elemen yang sama antara kedua himpunan tersebut). Untuk implementasi, digunakan fungsi utilitas untuk mendapatkan *value* dari suatu string atau line yang berisi transaksi dari file input

```
def parseTxnElmt(elmt: str): # Utilities function
    try:
        if(elmt[0] in ['R', 'W']):
            res = re.search('(^[1-9]\d*)+\+(\w)+\)', elmt[1:])
            txNum = res.group(1)
            data = res.group(2) # Handle only 1 char
            return elmt[0], int(txNum), data
        elif(elmt[0] == 'C'):
            return elmt[0], int(elmt[1:]), None
    except:
        print("Error occured")

def OCCProcedure():
    numTxn = None # Number of transaction
    dataTxn = [] # Data affected in all transactions (schedule)
    txnSequence = [] # Sequence of context switch between
transaction element
    validatedTxnId = [] # Transaction id that has been validated
    txn = [] # Array of transaction

    with open(file_name, 'r') as tx_file:
        # * Iterate through lines
```

```

for (idx, line) in enumerate(tx_file):
    if(idx == 0):
        numTxn = int(line)
        # Create transaction instance
        for i in range(numTxn):
            newTxn = OCCTransaction(i + 1)
            txn.append(newTxn)
    elif(idx == 1):
        data = line.split(' ')
        for val in data:
            dataTxn.append(val[0])
    else:
        txnSequence.append(line.rstrip('\n'))
        txnType, txnId, dataAffected =
Util.parseTxnElmt(line.rstrip('\n'))
        if(txnType == 'R'): # txnType
            txn[int(txnId) -
1].read_set.append(dataAffected) # Adding read_set
            elif(txnType == 'W'):
                txn[int(txnId) -
1].write_set.append(dataAffected) # Adding write_set

        start_time = time.time()
        for val in txnSequence:
            txnType, txnId, dataAffected = Util.parseTxnElmt(val)
            if(txnType in ['R', 'W']):
                # * Read phase
                if(txn[txnId - 1].START is None):
                    txnStart = round(time.time() - start_time, 2)
                    txn[txnId - 1].START = txnStart
                    print(f"[READ PHASE T{txnId}] Starting read phase
in transaction {txnId} in t={txn[txnId-1].START}")
                    isRead = True if txnType == 'R' else False

```



```

        read(dataAffected, isRead, txnId)
        time.sleep(1)
    elif(txnType == 'C'):
        # * Validation
        validated = False
        txn[txnId - 1].TS = round(time.time() - start_time, 2)
        print(f"[VALIDATION PHASE T{txnId}] Starting validation
of transaction {txnId}")
        if(not(validatedTxnId)): # If it is the first transaction
committed
            validated = True

        for val in (validatedTxnId):
            if(txn[val - 1].TS < txn[txnId - 1].TS):
                firstCondition = txn[val - 1].FINISH <
txn[txnId - 1].START
                isDisjoint = not(bool(set(txn[txnId -
1].read_set) & set(txn[val - 1].write_set)))
                secondCondition = (txn[val - 1].FINISH <
txn[txnId - 1].TS) and isDisjoint
                if(firstCondition or secondCondition):
                    validated = True
            time.sleep(1)

        # * Write phase
        # printTxnsTimestamp(txn) // # ! For debug

        if(validated):
            print(f"[VALIDATION PHASE T{txnId}] Finish
validating of T{txnId}")
        else: # * Exit failed case
            print(f"[VALIDATION PHASE T{txnId}] Failed to
validating T{txnId}")

```

```

        exit()

        print(f"[WRITE PHASE T{txnId}] Applied transaction
T{txnId} updates to the database")
        txn[txnId - 1].FINISH = round(time.time() - start_time,
2)

        validatedTxnId.append(txnId)
        time.sleep(1)

    # * Exit success case
    if(len(validatedTxnId) == numTxn):
        print("[!!!! FINISHED !!!!!] All transaction is
validated")

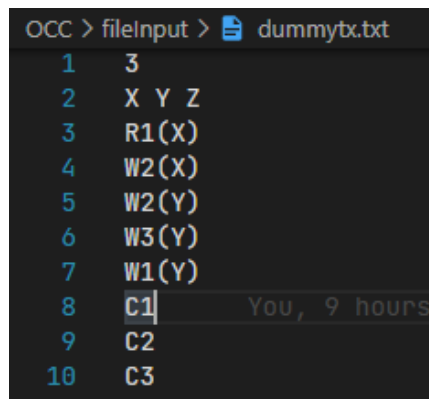
        exit()

```

### 1.2.2.3 Hasil percobaan

1. Kasus 1, dimana tidak terdapat konflik pada *schedule*

*Input:*



```

OCC > fileInput > dummytx.txt
1 3
2 X Y Z
3 R1(X)
4 W2(X)
5 W2(Y)
6 W3(Y)
7 W1(Y)
8 C1
9 C2
10 C3

```

*Output:*

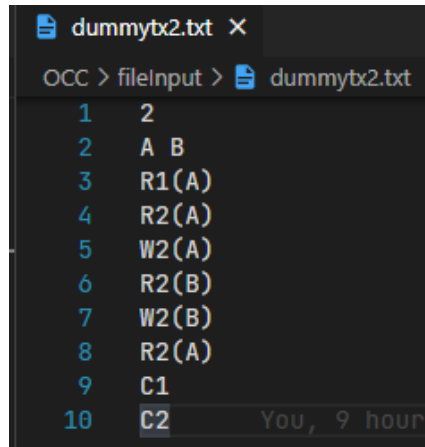
```

PS C:\Users\geral\Documents\IF_Semester_5_TA_2022-2023\IF3140 Manajemen Basis Data\Tugas\Tugas Besar 2\MBD02> python -u "c:\U
n Basis Data\Tugas\Tugas Besar 2\MBD02\OCC\OCCMain.py"
Masukan nama file: dummytx.txt
[READ PHASE T1] Starting read phase in transaction 1 in t=0.0
[READ PHASE T1] Read X from database and store it in local data version T1
[READ PHASE T2] Starting read phase in transaction 2 in t=1.01
[READ PHASE T2] Read X from database and store it in local data version T2, also write a value of X in local data version T2
[READ PHASE T2] Read Y from database and store it in local data version T2, also write a value of Y in local data version T2
[READ PHASE T3] Starting read phase in transaction 3 in t=3.02
[READ PHASE T3] Read Y from database and store it in local data version T3, also write a value of Y in local data version T3
[READ PHASE T1] Read Y from database and store it in local data version T1, also write a value of Y in local data version T1
[VALIDATION PHASE T1] Starting validation of transaction 1
[VALIDATION PHASE T1] Finish validating of T1
[WRITE PHASE T1] Applied transaction T1 updates to the database
[VALIDATION PHASE T2] Starting validation of transaction 2
[VALIDATION PHASE T2] Finish validating of T2
[WRITE PHASE T2] Applied transaction T2 updates to the database
[VALIDATION PHASE T3] Starting validation of transaction 3
[VALIDATION PHASE T3] Finish validating of T3
[WRITE PHASE T3] Applied transaction T3 updates to the database
[!!!! FINISHED !!!!] ALL transaction is validated

```

## 2. Kasus 2, dimana tidak terdapat konflik pada *schedule*

*Input:*



```

dummytx2.txt X
OCC > fileInput > dummytx2.txt
1 2
2 A B
3 R1(A)
4 R2(A)
5 W2(A)
6 R2(B)
7 W2(B)
8 R2(A)
9 C1
10 C2

```

*Output:*

```

PS C:\Users\geral\Documents\IF_Semester_5_TA_2022-2023\IF3140 Manajemen Basis Data\Tugas\Tugas Besar 2\MBD02> python -u "c:\U
n Basis Data\Tugas\Tugas Besar 2\MBD02\OCC\OCCMain.py"
Masukan nama file: dummytx2.txt
[READ PHASE T1] Starting read phase in transaction 1 in t=0.0
[READ PHASE T1] Read A from database and store it in local data version T1
[READ PHASE T2] Starting read phase in transaction 2 in t=1.0
[READ PHASE T2] Read A from database and store it in local data version T2
[READ PHASE T2] Read A from database and store it in local data version T2, also write a value of A in local data version T2
[READ PHASE T2] Read B from database and store it in local data version T2
[READ PHASE T2] Read B from database and store it in local data version T2, also write a value of B in local data version T2
[READ PHASE T2] Read A from database and store it in local data version T2
[VALIDATION PHASE T1] Starting validation of transaction 1
[VALIDATION PHASE T1] Finish validating of T1
[WRITE PHASE T1] Applied transaction T1 updates to the database
[VALIDATION PHASE T2] Starting validation of transaction 2
[VALIDATION PHASE T2] Finish validating of T2
[WRITE PHASE T2] Applied transaction T2 updates to the database
[!!!! FINISHED !!!!] ALL transaction is validated

```

## 3. Kasus 3, dimana terdapat konflik pada *schedule*

*Input:*

```
dummytx3.txt X
OCC > fileInput > dummytx3.txt
You, 9 hours ago | 1 author (You)
1 2
2 A B
3 R1(A)
4 W1(A)
5 R2(A)
6 W2(A)
7 R2(B)
8 W2(B)
9 R2(A)
10 C1 You, 9 hours a
11 C2
```

Output:

```
PS C:\Users\geral\Documents\IF_Semester_5_TA_2022-2023\IF3140 Manajemen Basis Data\Tugas\Tugas Besar 2\MBD02> python -u "c:\Us
n Basis Data\Tugas\Tugas Besar 2\MBD02\OCC\OCCMain.py"
Masukan nama file: dummytx3.txt
[READ PHASE T1] Starting read phase in transaction 1 in t=0.0
[READ PHASE T1] Read A from database and store it in local data version T1
[READ PHASE T1] Read A from database and store it in local data version T1, also write a value of A in local data version T1
[READ PHASE T2] Starting read phase in transaction 2 in t=2.01
[READ PHASE T2] Read A from database and store it in local data version T2
[READ PHASE T2] Read A from database and store it in local data version T2, also write a value of A in local data version T2
[READ PHASE T2] Read B from database and store it in local data version T2
[READ PHASE T2] Read B from database and store it in local data version T2, also write a value of B in local data version T2
[READ PHASE T2] Read A from database and store it in local data version T2
[VALIDATION PHASE T1] Starting validation of transaction 1
[VALIDATION PHASE T1] Finish validating of T1
[WRITE PHASE T1] Applied transaction T1 updates to the database
[VALIDATION PHASE T2] Starting validation of transaction 2
[VALIDATION PHASE T2] Failed to validating T2
```

## 1.3 Eksplorasi *Recovery*

### 1.3.1 *Write-Ahead Log*

*Write-Ahead Log* atau WAL adalah metode *log based recovery* yang bertujuan untuk menjamin *atomicity* dan *durability* dari basis data. *Atomicity* adalah karakter basis data yang akan melihat eksekusi-eksekusi pada satu transaksi yang sama adalah satu kesatuan. Sehingga, jika terdapat eksekusi yang gagal pada suatu transaksi, eksekusi-eksekusi sebelumnya yang berhasil dilakukan akan di-*rollback*. Karakteristik ini menjamin semua eksekusi transaksi akan sukses dijalankan semua atau tidak sama sekali. Sedangkan, *durability* adalah karakteristik basis data yang memastikan bahwa semua data hasil eksekusi transaksi akan tersimpan secara permanen sehingga tidak akan hilang. Hal ini dapat dilakukan dengan menyimpan data-data tersebut pada memori *nonvolatile*.

*Write-Ahead Log recovery* digunakan untuk menangani terjadinya *clash* atau kegagalan transaksi ketika kondisi log transaksi hanya berada di *main memory* dan belum dituliskan ke *stable storage*. Apabila log berada pada *main memory* dan kemudian mengalami sistem mengalami *clash*, maka log tersebut akan menghilang.

Aturan yang harus dipenuhi untuk melakukan *Write-Ahead Log recovery* ini adalah:

1. Transaksi  $T_i$  berada pada *commit state* hanya jika *log record*  $\langle T_i, \text{commit} \rangle$  sudah ditulis ke *stable storage*
2. Sebelum transaksi  $T_i$  ditulis ke *stable storage*, transaksi lain yang berkaitan dengan transaksi  $T_i$  harus ditulis ke *stable storage*
3. Sebelum blok data hasil transaksi  $T_i$  pada *main memory* ditulis ke basis data atau *nonvolatile storage*, blok data lain yang berhubungan dengan blok data transaksi  $T_i$  tersebut harus ditulis ke *stable storage*

Dengan WAL, apabila terjadi perubahan data akibat suatu transaksi, setiap *log record* tidak perlu untuk ditulis ke *stable storage*. Hal ini menyebabkan pengurangan jumlah *disk writes* secara signifikan. Selain itu, *cost* yang diperlukan untuk sinkronisasi juga akan berkurang karena log yang ditulis pada file WAL akan ditulis secara terurut. Kemudian, sekumpulan file WAL yang telah terbentuk tersebut akan digunakan pada *continuous archiving* untuk melakukan *backup* selanjutnya secara berkelanjutan.

### 1.3.2 Continuous Archiving

*Continuous Archiving* adalah mekanisme *recovery* yang dilakukan secara berkelanjutan. Untuk melakukan *continuous archiving*, terdapat dua bagian yang harus dilakukan, yaitu *base backup* dan *WAL archive*. *Base backup* dilakukan dengan membuat salinan dari basis data secara keseluruhan pada suatu waktu tertentu. Setelah *base backup* telah dibuat, hasil tersebut akan dalam bentuk *WAL file* yang berisi log-log eksekusi transaksi secara sekuensial. *WAL file* akan dikelompokkan menjadi suatu list yang disebut sebagai *WAL archive list*.

Untuk melakukan *continuous archiving*, dapat dilakukan dengan melakukan perintah `pg_basebackup -Ft -D <directory>`. *Command* ini akan membuat *base backup* ke suatu *directory*. Setelah melakukan *base backup*, perlu dilakukan konfigurasi *WAL archive*. Hal ini

dilakukan agar untuk setiap file WAL yang dibuat, *command archive* pada konfigurasi tersebut akan selalu dieksekusi. *Continuous archiving* juga dilakukan agar *point in time recovery* dapat dilakukan.

### 1.3.3 Point in Time Recovery

*Point in Time Recovery* adalah metode *recovery* yang dilakukan ke suatu kondisi basis data pada titik waktu manapun secara spesifik. Untuk melakukan *Point in Time Recovery*, diperlukan *continuous archiving* terlebih dahulu. Semua *backup file* yang ada pada kurun waktu sebelum waktu pengguna ingin melakukan *backup* harus ada. *Backup* tersebut harus ditulis ke *stable storage*. Apabila terdapat *backup file* yang kurang atau *transaction log* yang terpotong, *Point in Time Recovery* tidak dapat dilakukan.

*Point in Time Recovery* dilakukan dengan melakukan restorasi semua WAL *file* yang terbuat dalam kurun waktu *timestamp* kurang dari waktu yang pengguna inginkan secara sekuensial. Untuk melakukan restorasi, pastikan terlebih dahulu semua *backup file* yang sudah terbuat dan ada di PostgreSQL. Sebelum memulai restorasi, matikan server terlebih dahulu. Kemudian, jalankan perintah `pg-basebackup` untuk menyalin file-file *backup* yang telah di-generate sebelumnya sehingga *base backup* akan otomatis terbaca dan men-trigger restorasi pada PostgreSQL.

Setelah *base backup* telah berhasil direstorasi, akan dilakukan restorasi log-log yang terdapat pada WAL *file*. Hal ini dilakukan dengan membuat file *recovery.conf* kemudian menuliskan *recovery\_target\_time* dan *restore\_command*. Pada *recovery\_target\_time* akan memuat informasi terkait *timestamp* waktu basis data yang ingin direstorasi. Sedangkan *restore\_command* akan digunakan untuk menyalin WAL *file* yang terdapat pada WAL *archive*.

Setelah itu, jalankan server kembali. PostgreSQL akan membaca adanya *base backup* dan file *recovery.conf* yang sudah diatur sebelumnya. Sehingga basis data akan ter-trigger untuk melakukan restorasi dengan perintah *restore command* yang sudah diatur sebelumnya pada file konfigurasi dengan basis data yang akan di-restore sebelum kurun waktu yang diatur pada *recovery\_target\_time* yang sudah diatur sebelumnya pada file konfigurasi. Sehingga kondisi basis data akan direstorasi sesuai dengan kondisi basis data pada *timestamp* tersebut.

### 1.3.4 Simulasi Kegagalan

1. Sebelum melakukan simulasi kegagalan, akan dibuat *directory* baru yang nantinya akan menjadi tempat penyimpanan hasil arsip file WAL ketika melakukan *continuous archiving*. *Directory* akan dibuat pada path `/var/lib/postgresql/pg_log_archive` dengan menggunakan *command* berikut.

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

```
hildeward@hildeward-VivoBook-ASUSLaptop-X409JP-A409JP:~$ sudo -H -u postgres mkdir /var/lib/postgresql/pg_log_archive
hildeward@hildeward-VivoBook-ASUSLaptop-X409JP-A409JP:~$ ls /var/lib/postgresql
12  pg_log_archive
```

2. Selanjutnya, akan dilakukan konfigurasi untuk mengaktifkan WAL *logging* dan aturan *recovery* ketika melakukan *continuous archiving*. Konfigurasi dilakukan pada file `postgresql.conf` yang terdapat pada `/etc/postgresql/12/main/postgresql.conf`.

*Command* untuk mengubah konfigurasi file adalah sebagai berikut.

```
$ sudo nano /etc/postgresql/12/main/postgresql.conf
```

Konfigurasi dilakukan dengan menambahkan aturan berikut ini.

```
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'
```

```
# - Settings -
```

```
wal_level = replica                                # minimal, replica, or logical
```

```
# - Archiving -
```

```
archive_mode = on                                # enables archiving; off, on, or always
                                                # (change requires restart)
#archive_library = ''                            # library to use to archive a logfile segment
                                                # (empty string indicates archive_command should
                                                # be used)
archive_command = 'test ! -f /var/lib/postgresql/pg_log_archive/%f && cp %p /var/lib/postgresql/pg_log_archive/%f'
```

3. Agar konfigurasi yang telah dilakukan sebelumnya dapat teraplikasikan, DBMS perlu di-*restart* terlebih dahulu dengan menggunakan *command* berikut.

```
$ sudo systemctl restart postgresql@15-main
$ sudo systemctl restart postgresql
```

4. Untuk melakukan pengetesan simulasi kegagalan, akan dibuat basis data baru dan tabel baru.

Menggunakan *role user* postgres dalam pembuatan basis data dan tabel.

```
$ sudo -u postgres -i
$ psql -c "CREATE DATABASE tubes2mbd;"
$ psql tubes2mbd -c "CREATE TABLE matkulIF (ID integer, nama character
varying(100));"
```

Membuat *database* tubes2mbd dan *table* matkulIF.

```
$ sudo -u postgres -i
$ psql -c "CREATE DATABASE tubes2mbd;"
$ psql tubes2mbd -c "CREATE TABLE matkulIF (ID integer, nama character
varying(100));"
```

```
hildegard@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ sudo -u postgres -i
postgres@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ psql -c "CREATE DATABASE tubes2mbd;"
CREATE DATABASE
postgres@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ psql tubes2mbd -c "CREATE TABLE matkulIF (ID integer, nama character varying(100));"
CREATE TABLE
```

5. Membuat *backup* kluster basis data yang telah dibuat hingga saat ini ke *directory* /var/lib/postgresql dengan format file .tar.gz. *Backup* kluster ini akan digunakan nantinya untuk melakukan *recovery*.

*Command* yang digunakan untuk melakukan *backup* adalah pg\_basebackup dengan penggunaan seperti berikut.

```
$ pg_basebackup -Ft -X none -D - | gzip >
/var/lib/postgresql/db_file_backup.tar.gz
```

Untuk mengecek apakah *backup* telah berhasil dilakukan, dapat melihat isi dari folder /var/lib/postgresql dan melihat apakah file db\_file\_backup.tar.gz tersedia.



```
postgres@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ pg_basebackup -Ft -X none -D - | gzip > /var/lib/postgresql/db_file_backup.tar.gz
NOTICE: all required WAL segments have been archived
postgres@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ ls /var/lib/postgresql
12 db_file_backup.tar.gz pg_log_archive
```

6. Selanjutnya, akan dimasukkan data ke tabel pada *database* yang telah dibuat pada poin 4.

*Command* yang digunakan adalah sebagai berikut.

```
$ psql tubes2mbd -c
"INSERT INTO matkulIF (id, nama)
VALUES (3140, 'Manajemen Basis Data'),
      (3130, 'Jaringan Komputer'),
      (3140, 'Inteligensia Buatan');"
```

Akan dilakukan pengecekan apakah data berhasil dimasukkan dengan melihat isi tabel.

```
$ psql tubes2mbd -c "SELECT * FROM matkulIF"
```

```
postgres@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ psql tubes2mbd -c "INSERT INTO matkulIF (id, nama) VALUES (3140,
'Manajemen Basis Data'), (3130, 'Jaringan Komputer'), (3140, 'Inteligensia Buatan');"
INSERT 0 3
postgres@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ psql tubes2mbd -c "SELECT * FROM matkulIF;"
 id |      nama
-----+-----
 3140 | Manajemen Basis Data
 3130 | Jaringan Komputer
 3140 | Inteligensia Buatan
(3 rows)
```

Kemudian, akan dilakukan pencatatan *timestamp* dengan menggunakan *command* `select now()` sebagai berikut.

```
$ psql -c "select now()"
```

```
postgres@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ psql -c "select now()"
      now
-----
2022-11-29 20:30:33.914804+07
(1 row)
```

7. Selanjutnya, akan dilakukan perubahan data pada *database* dan tabel yang telah dibuat sebelumnya. Tujuan dilakukan pembaharuan data adalah untuk mengetahui data apa yang di-*recovery* nantinya, apakah data terbaru atau data pada *recovery target time* tertentu.

*Command* yang digunakan untuk meng-*update* data adalah sebagai berikut.

```
$ psql tubes2mbd -c "UPDATE matkulIF SET nama = 'dummy' WHERE ID = 3140;"
```

Akan dilakukan pengecekan isi tabel data juga dengan melihat semua data pada tabel.

```
postgres@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ psql tubes2mbd -c "UPDATE matkulIF SET nama = 'dummy' WHERE ID = 3140;"
UPDATE 2
postgres@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ psql tubes2mbd -c "SELECT * FROM matkulIF;"
 id |      nama
-----+-----
 3130 | Jaringan Komputer
 3140 | dummy
 3140 | dummy
(3 rows)
```

8. Melakukan pengarsipan *log* WAL. Untuk menjamin pengarsipan *log* WAL dilakukan, akan dilakukan dengan memaksa DBMS untuk mengarsipkan WAL *file* dari basis data dengan menggunakan *command* `pg_switch_wal()` sebagai berikut.

```
$ psql -c "select pg_switch_wal();"
```

```
postgres@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ psql -c "select pg_switch_wal();"
pg_switch_wal
-----
0/6000490
(1 row)
```

9. Simulasi kegagalan akan dilakukan dengan mematikan *server* DBMS dan menghapus semua konten pada basis data yang terdapat pada folder `/var/lib/postgresql/10/main/`.

*Command* yang digunakan untuk mematikan dan menghapus basis data adalah sebagai berikut.

```
$ sudo systemctl stop postgresql
$ rm /var/lib/postgresql/10/main/* -r
```

```
hildegard@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ sudo rm /var/lib/postgresql/12/main -r
hildegard@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ sudo ls /var/lib/postgresql/12/main
ls: cannot access '/var/lib/postgresql/12/main': No such file or directory
hildegard@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ sudo mkdir /var/lib/postgresql/12/main
hildegard@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ sudo ls /var/lib/postgresql/12/main
hildegard@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ sudo ls /var/lib/postgresql/12
main
```

Apabila mencoba untuk mengakses basis data, akan didapatkan pesan *error* sebagai berikut.

```
postgres@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ psql tubes2mbd -c "SELECT * FROM matkulIF;"
psql: error: could not connect to server: No such file or directory
        Is the server running locally and accepting
        connections on Unix domain socket "/var/run/postgresql/.s.PGSQL.5432"?
```

10. *Recovery* akan dilakukan dengan memanfaatkan data *backup* yang sudah dibuat sebelumnya, yaitu *db\_file\_backup.tar.gz* dan WAL file yang telah tersimpan pada *directory* yang dibuat pada poin pertama.

*Command* yang digunakan untuk mengembalikan data pada folder */var/lib/postgresql/12/main/* dari *backup* file adalah sebagai berikut.

```
$ tar xvfz /var/lib/postgresql/db_file_backup.tar.gz -C
/var/lib/postgresql/12/main/
```

```
hildegard@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ sudo tar xvfz /var/lib/postgresql/db_file_backup.tar.gz -C /var/lib/postgresql/12/main/
backup_label
tablespace_map
PG_VERSION
pg_logical/
pg_logical/mappings/
pg_logical/replorigin_checkpoint
pg_logical/snapshots/
pg_multixact/
pg_multixact/offsets/
pg_multixact/offsets/0000
pg_multixact/members/
pg_multixact/members/0000
global/
global/2964
global/1262_fsm
global/4176
global/6001
global/1261_vm
global/1262
global/1260_fsm
global/6114
global/2677
global/2965
global/2396_vm
global/6100
global/4183
global/4178
global/4175
global/1232
global/1136
global/1233
global/4182
global/1262_vm
global/1260
```

Sehingga, apabila melakukan pengecekan isi dari folder */var/lib/postgresql/12/main/*, akan terdapat file dan folder hasil *backup* dari *command* sebelumnya.

```
hildegard@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ sudo ls /var/lib/postgresql/12/main
backup_label  global          pg_dynshmem    pg_multixact   pg_replslot    pg_snapshots   pg_stat_tmp    pg_tblspc      PG_VERSION     pg_xact         tablespace_map
base          pg_commit_ts   pg_logical     pg_notify      pg_serial      pg_stat        pg_subtrans    pg_twophase    pg_wal         postgresql.auto.conf
```

11. Selanjutnya, untuk melakukan *recovery* ke suatu poin waktu tertentu, akan dibuat suatu file konfigurasi *recovery* pada folder */var/lib/postgresql/10/main/*. File *recovery* ini akan berisi *restore\_command* dan *recovery\_target\_time* untuk menspesifikan data pada poin waktu berapa yang akan di-*recovery*.

Command yang digunakan adalah sebagai berikut.

```
$ sudo nano /var/lib/postgresql/10/main/recovery.conf
```

Isi dari file recovery.conf adalah sebagai berikut.

```
restore_command = 'cp /var/lib/postgresql/pg_log_archive/%f %p'
recovery_target_time = '2022-11-29 20:30:33.914804+07'
```

```
GNU nano 4.8 /var/lib/postgresql/12/main/recovery.conf Modified
restore_command = 'cp /var/lib/postgresql/pg_log_archive/%f %p'
recovery_target_time = '2022-11-29 20:30:33.914804+07'
```

12. Selanjutnya, nyalakan kembali DBMS dan lakukan verifikasi keberhasilan *recovery* dengan melakukan pengecekan 20 *log* terakhir.

Command yang digunakan untuk menyalakan DBMS adalah sebagai berikut.

```
$ sudo systemctl start postgresql
```

Command yang digunakan untuk mengecek *log* basis data adalah sebagai berikut.

```
$ tail -n 10 /var/log/postgresql/postgresql-12-main.log
```

```
hildegard@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ tail -n 20 /var/log/postgresql/postgresql-12-main.log
2022-11-29 20:15:58.994 WIB [5131] LOG: database system is ready to accept connections
2022-11-29 20:20:20.825 WIB [5131] LOG: received fast shutdown request
2022-11-29 20:20:20.944 WIB [5131] LOG: aborting any active transactions
2022-11-29 20:20:20.948 WIB [5131] LOG: background worker "logical replication launcher" (PID 5138) exited with exit code 1
2022-11-29 20:20:20.948 WIB [5133] LOG: shutting down
2022-11-29 20:20:20.948 WIB [5133] LOG: database system is shut down
2022-11-29 20:20:29.311 WIB [5553] LOG: starting PostgreSQL 12.12 (Ubuntu 12.12-0ubuntu0.20.04.1) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 9.4.0-1ubuntu1-20.04.1) 9.4.0, 64-bit
2022-11-29 20:20:29.344 WIB [5553] LOG: listening on IPv4 address "127.0.0.1", port 5432
2022-11-29 20:20:29.344 WIB [5553] LOG: listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2022-11-29 20:20:29.461 WIB [5554] LOG: database system was shut down at 2022-11-29 20:20:29 WIB
2022-11-29 20:20:29.607 WIB [5553] LOG: database system is ready to accept connections
2022-11-29 20:20:41.297 WIB [5584] hildegard@hildegard FATAL: role "hildegard" does not exist
2022-11-29 20:20:55.823 WIB [5600] hildegard@hildegard FATAL: role "hildegard" does not exist
2022-11-29 20:28:29.910 WIB [5956] postgres@tubes2mbd ERROR: unterminated quoted string at or near "'Inteligensia Buatan';" at character 108
2022-11-29 20:28:29.910 WIB [5956] postgres@tubes2mbd STATEMENT: INSERT INTO matkulIF (id, nama) VALUES (3140, 'Manajemen Basis Data'), (3130, 'Jaringan Komputer'), (3140, 'Inteligensia Buatan');
2022-11-29 20:35:00.126 WIB [5553] LOG: received fast shutdown request
2022-11-29 20:35:00.241 WIB [5553] LOG: aborting any active transactions
2022-11-29 20:35:00.243 WIB [5553] LOG: background worker "logical replication launcher" (PID 5561) exited with exit code 1
2022-11-29 20:35:00.244 WIB [5555] LOG: shutting down
2022-11-29 20:35:01.131 WIB [5553] LOG: database system is shut down
```

13. Terakhir, akan dilakukan pengecekan kembali isi dari tabel basis data yang telah dibuat.

```
postgres@hildegard-VivoBook-ASUSLaptop-X409JP-A409JP:~$ psql tubes2mbd -c "SELECT * FROM matkulIF;"
 id |      nama
-----+-----
 3140 | Manajemen Basis Data
 3130 | Jaringan Komputer
 3140 | Inteligencia Buatan
(3 rows)
```

Berdasarkan hasil tersebut, terlihat bahwa data yang di-*recovery* adalah data-data yang ada pada waktu sebelum *recovery target time*. Data yang di luar dari kurun waktu tersebut tidak akan di-*recovery* oleh basis data, seperti data hasil peng-*update*-an pada poin 7.

## Bab 2

### Kesimpulan dan Saran

#### 2.1 Kesimpulan

Eksekusi *simple locking* dapat dilakukan dengan cepat dikarenakan algoritmanya yang tidak terlalu sulit. Lalu, untuk implementasi *optimistic concurrency protocol* dapat dilakukan pada beberapa kasus uji dan sebagian tidak mengingat kasus dimana terjadi konflik pada transaksi menyebabkan transaksi tidak dapat dilakukan. Derajat Isolasi yang paling aman adalah *Serializability* karena mencegah terjadinya fenomena yang tidak seharusnya terjadi pada basis data sehingga konsep ACID tetap terjaga.

#### 2.2 Saran

Pada implementasi OCC (*Optimistic concurrency protocol*) dapat dilakukan pengembangan lagi dari segi *parsing* data input agar dapat meng-*handle* data transaksi dengan lebih dari satu karakter dan meng-*handle id* transaksi yang tidak harus sekuensial. Implementasi algoritma ini juga dapat dilakukan dengan paradigma lain, seperti *object-oriented programming* untuk memudahkan keterbacaan algoritma.

## Pembagian Kerja

NIM	Nama	Konten kerja
13520012	Aji Andhika Falah	- Bagian implementasi <i>simple lock</i>
13520063	Louis Yanggara	- Bagian eksplorasi <i>concurrency control</i>
13520184	Adelline Kania Setiyawan	- Bagian eksplorasi <i>recovery</i>
13520138	Gerald Abraham Sianturi	- Bagian implementasi SOCC

## Referensi

- [1] Silberschatz et al. 2011. Database System Concept, 6th Ed. New York: McGraw-Hill.
- [2] <https://mkdev.me/posts/transaction-isolation-levels-with-postgresql-as-an-example>
- [3] <https://www.postgresql.org/docs/current/transaction-iso.html#MVCC-ISOLEVEL-TABLE>