

The background of the slide is a repeating pattern of green tropical leaves, specifically palm fronds, on a white background. The leaves are scattered across the entire frame, with some appearing more prominently than others.

# **TRANSAKSI DATABASE**

**TIM MBD**

# TRANSAKSI

- *Sistem database secara normal adalah diakses oleh banyak user atau proses pada waktu yang bersamaan.*
  - ❖ *Baik Query maupun Modifikasi (insert, update, delete)*
- *Tidak seperti sistem operasi (seperti: Windows atau Linux) yang memungkinkan adanya interaksi diantara proses, sebuah DBMS perlu untuk menjaga proses-proses didalamnya untuk terhindar dari masalah yang mungkin timbul dari adanya interaksi antar proses.*

# TRANSAKSI

- ***Transaksi** = adalah proses-proses yang melibatkan query dan / atau modifikasi pada database*
- *Umumnya dengan beberapa property yang berkaitan dengan masalah concurrency*
- *Terbentuk dalam SQL dari statemen tunggal atau kontrol programmer secara eksplisit*

# ACID Transaction

## A. ATOMICITY

Memastikan bahwa keseluruhan operasi suatu transaksi dikerjakan seluruhnya atau tidak sama sekali.



**Account Jasmine**  
Rp. 1.000.000,-

**Operasi Pertama** : Pengurangan  
saldo Rp 500.000,-

Jasmine transfer Rp 500.000,- ke Ariel

Konsep **Atomicity** memastikan hal ini tidak terjadi



**Account Ariel**  
Rp. 1.000.000,-

**Operasi Kedua** : Penambahan  
saldo Rp 500.000,-

*Seharusnya saldo  
akun WH Rp  
1.500.000,-*



# ACID Transaction

## B. CONSISTENT

Mempertahankan batasan-batasan pada database, seperti memastikan data yang ditulis harus valid sesuai dengan semua aturan yang diberikan.

# ACID Transaction

## C. ISOLATED

Memastikan dalam suatu waktu hanya ada satu proses yang melibatkan suatu *user* (menghindari proses yang berjalan paralel).

### Transaksi Berjalan Paralel



Account Jasmine  
Rp. 1.800.000,-

Jasmine menabung  
Rp. 500.000,-

Transaksi 1

Transaksi 2

Konsep **Isolated** memastikan hal ini tidak terjadi

Data

Rp 1.500.000,-

Rp 1.800.000,-

Karena paralel, maka transaksi 2 membaca akun Jasmine sebelum ada perubahan dari transaksi 1 sehingga saldo Jasmine yang seharusnya **Rp 2.300.000,-** tertulis **Rp 1.800.000,-**

# ACID Transaction

## D. DURABILITY

Memastikan perubahan apapun yang sudah di-*commit* pada *database* harus tetap terjadi pada *database* tersebut sekalipun ada *crash*.

Contoh : Apabila saldo akun PJW adalah Rp 1.500.000,- maka apabila ada *crash*, database harus tetap menyimpan saldo PJW adalah Rp 1.500.000,-

# COMMIT

- *Statement COMMIT pada SQL menyebabkan sebuah transaksi selesai sempurna*
- *Setiap modifikasi pada database akan tersimpan secara permanen*

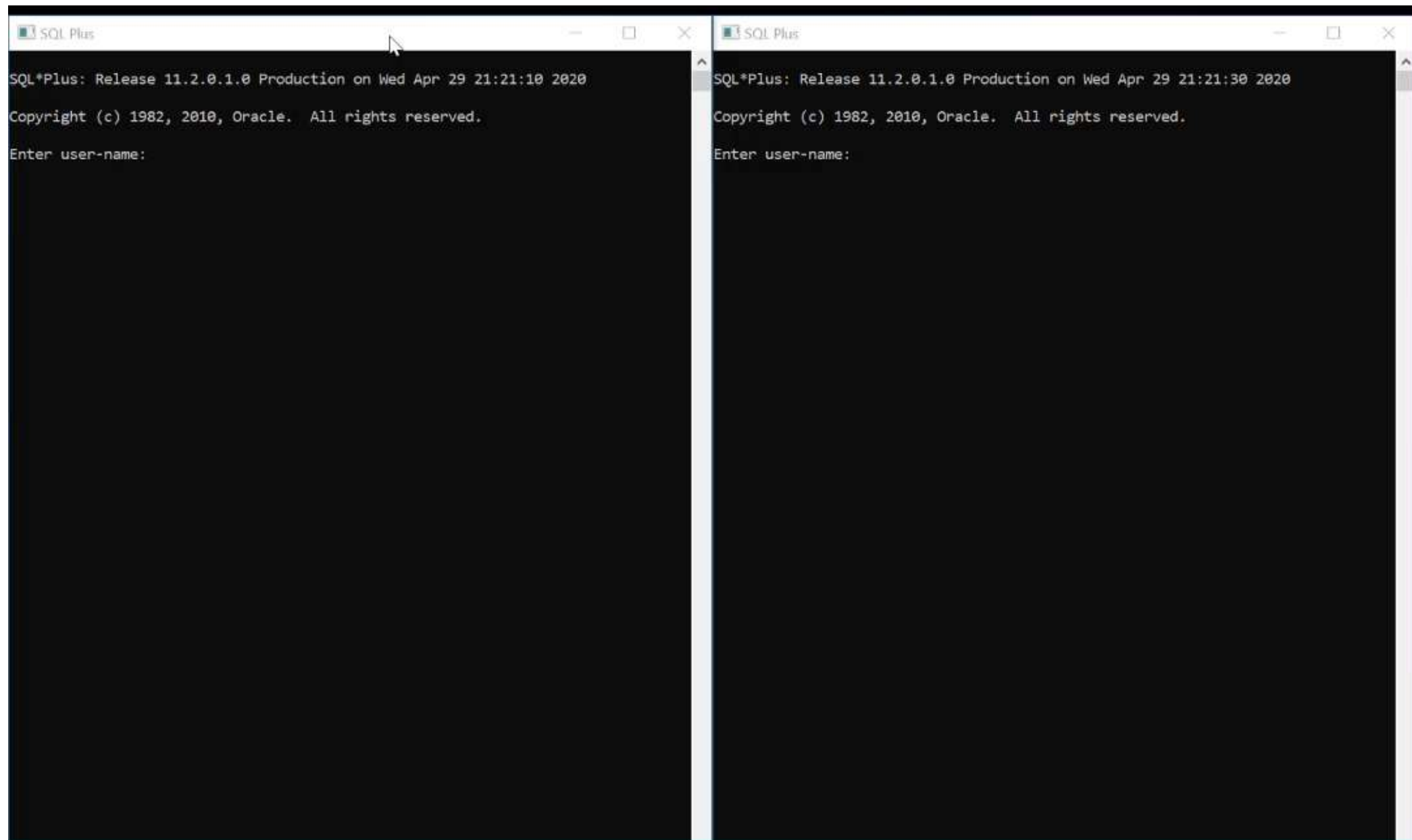


# ROLLBACK

*Statement Rollback pada SQL juga menyebabkan sebuah transaksi selesai, tetapi dengan cara menggagalkan (abort)*

- *Tidak menimbulkan efek apa-apa pada database*

*Kesalahan seperti pembagian dengan 0 atau constraint violation (pelanggaran terhadap batasan) dapat juga menyebabkan proses ROLLBACK, bahkan jika sang programmer tidak memintanya*



Membuka Dua Layar SQLPlus dengan user sysdba dimana Layar Kanan dan Kiri sama-sama melakukan SELECT pada Tabel OBAT

# BAGAIMANA PROSES BERINTERAKSI?

Obat				
Obat_Id	Obat_Nama	Obat_Stok	Obat_Satuan	Obat_Harga
B001	Obat A	50	strip	2000
B002	Obat B	250	Kapsul	1000
B003	Obat C	300	tablet	2000
B004	Obat D	500	botol	25000

1. *Elsa ingin mengetahui jumlah stok tertinggi dan stok terendah obat*
2. *Anna mengubah seluruh stok obat menjadi 600*

# BAGAIMANA PROSES BERINTERAKSI?

## *Program Elsa*

*Elsa mengeksekusi dua statemen SQL untuk menampilkan obat dengan stok maksimal dan minimal.*

```
SELECT MAX (Obat_Stok) FROM Obat;
```

```
SELECT MIN (Obat_Stok) FROM Obat;
```

# CONTOH : PROSES BERINTERAKSI

## *Program Anna*

*Anna mengeksekusi satu SQL untuk mengupdate stok semua obat menjadi 600*

```
UPDATE Obat SET Obat_Stok = 600;
```

# INTERLEAVING OF STATEMENT

*Sekalipun jika dilihat query MIN akan berjalan setelah query MAX, akan tetapi tidak ada batasan untuk menjalankan query pada program Anna atau Elsa terlebih dahulu.*

*Contoh : misalkan urutan eksekusi adalah **MAX – UPDATE - MIN***

Statement SQL	Hasil
MAX	500
UPDATE	Semua stok obat menjadi 600
MIN	600

*Dari eksekusi berikut, Elsa mendapat nilai max < min dikarenakan interleaving statement*

# INTERLEAVING OF STATEMENT

## *Solusi Interleaving of Statement*

*Statement dari Elsa dimasukkan ke dalam satu transaksi, maka Elsa tidak akan dapat melihat inkonsistensi dan dapat menampilkan nilai **max** dan **min** dengan menggunakan data yang sama*

# ROLLBACK

1. *Anna mengeksekusi sql **UPDATE** bukan sebagai transaksi, kemudian melakukan **ROLLBACK** untuk mengembalikan data karena dirasa update yang dilakukan tidak tepat.*
2. *Jika Elsa mengeksekusi statement min setelah **UPDATE** dan sebelum **ROLLBACK**, maka Elsa akan mendapat nilai yang di-update dimana pada kenyataannya nilai tersebut tidak ada di database*



# SOLUSI

- *Jika Anna mengeksekusi **UPDATE** sebagai transaksi, maka orang lain tidak dapat melihat yang dilakukan Anna sebelum Anna melakukan **COMMIT***
- *Apabila Anna melakukan **ROLLBACK** pada transaksi **UPDATE**, maka akibat dari transaksi tersebut tidak akan pernah terlihat*

# ISOLATION LEVEL

- *SQL mendefinisikan 4 level isolasi = pilihan interaksi yang diijinkan oleh transaksi yang dijalankan pada waktu yang hampir bersamaan.*
- *Setiap DBMS mengimplementasikan transaksi dengan caranya masing-masing*

# ISOLATION LEVEL

*Standar ANSI / ISO SQL mendefinisikan empat tingkat isolasi transaksi, dengan hasil yang berbeda untuk skenario transaksi yang sama*

- 1. SERIALIZABLE*
- 2. REPEATABLE READ*
- 3. READ COMMITTED*
- 4. READ UNCOMMITTED*

# ISOLATION LEVEL

*Isolation level didefinisikan dalam 3 hal :*

- 1. Dirty Reads*
- 2. Non-Repeatable Reads*
- 3. Phantom Reads*

# ISOLATION LEVEL

## ***Dirty Reads***

*Dirty Reads terjadi apabila ada transaksi yang dibaca oleh transaksi lain sebelum transaksi tersebut di-commit*

### **Transaksi 1**

```
SELECT  Obat_Stok  FROM
Obat WHERE  Obat_Nama  =
'Obat A' ;
```

```
SELECT  Obat_Stok  FROM
Obat WHERE  Obat_Nama  =
'Obat A' ;
```

### **Transaksi 2**

```
UPDATE      Obat      SET
Obat_Stok  =  600  WHERE
Obat_Nama  =  'Obat A' ;
```

```
ROLLBACK ;
```

Query Select Ke-2 dari Transaksi 1 melihat Query Transaksi 2 yang **belum di-commit**, sehingga query ke-2 menghasilkan **600** padahal angka tersebut tidak ada di database. Ini yang disebut **dirty reads**

# ISOLATION LEVEL

## ***Non-Repeatable Reads***

*Non-Repeatable Reads terjadi apabila selama dalam transaksi terdapat suatu baris yang dieksekusi dua kali akan tetapi nilai dalam baris tersebut berbeda dalam pembacaan*

### **Transaksi 1**

```
SELECT Obat_Stok FROM  
Obat WHERE Obat_Nama =  
'Obat A';
```

```
SELECT Obat_Stok FROM  
Obat WHERE Obat_Nama =  
'Obat A';  
COMMIT;
```

### **Transaksi 2**

```
UPDATE Obat SET  
Obat_Stok = 600 WHERE  
Obat_Nama = 'Obat A';  
COMMIT;
```

Pada beberapa isolasi level, Query ke-2 dari Transaksi 1 akan menampilkan nilai **600** dimana Query ke-1 menampilkan nilai **50** (nilai stok Obat A sebelum UPDATE). Hal ini disebut **Non-Repeatable Read**

# ISOLATION LEVEL

## *Phantom Reads*

*Phantom Reads terjadi apabila selama dalam transaksi terdapat suatu baris yang ditambahkan atau dihilangkan pada saat data sedang dibaca*

### Transaksi 1

```
SELECT Obat_Stok FROM  
Obat WHERE Obat_Stok  
BETWEEN 250 AND 500;
```

```
SELECT Obat_Stok FROM  
Obat WHERE Obat_Stok  
BETWEEN 250 AND 500;  
COMMIT;
```

### Transaksi 2

```
INSERT INTO Obat VALUES  
( 'B005', 'Obat E', 300,  
'tablet', 25000 );  
COMMIT;
```

Apabila Query ke-2 dari Transaksi 1 menampilkan daftar stok yang telah ditambahkan data dari Transaksi 2, maka hasil dari Query ke-2 dan Query ke-1 pada Transaksi 1 akan berbeda. Ini dinamakan **Phantom Reads**.

# ISOLATION LEVEL

	Dirty Reads	Non-Repeatable Reads	Phantom Reads
<b>Read Uncommitted</b>	Terjadi	Terjadi	Terjadi
<b>Read Committed</b>	Tidak terjadi	Terjadi	Terjadi
<b>Repeatable Read</b>	Tidak terjadi	Tidak terjadi	Terjadi
<b>Serializable</b>	Tidak terjadi	Tidak terjadi	Tidak terjadi



# READ UNCOMMITTED

**READ UNCOMMITTED :**  
*dapat melihat data didalam database, bahkan jika data tersebut ditulis oleh transaksi yang belum di-commit*


*Contoh dalam SQL SERVER dimana Query Transaksi 2 dijalankan setelah Query Transaksi 1*

## Transaksi 1

```
begin tran
update Obat set
Obat_Stok = 600 where
Obat_Nama = 'Obat A'
waitfor delay '00:00:20'
rollback
```

## Transaksi 2

```
set transaction
isolation level
read uncommitted
select Obat_Stok
from Obat where
Obat_Nama = 'Obat
A'
```



Hasil adalah **600** karena select pada Transaksi 2 akan berjalan sebelum rollback pada Transaksi 1.

# READ COMMITTED

**READ COMMITTED** : dapat melihat hanya data-data yang sudah di commit saja, tetapi bisa jadi datanya akan berubah pada waktu yang berbeda


Contoh dalam SQL SERVER dimana Query Transaksi 2 dijalankan setelah Query Transaksi 1

## Transaksi 1

```
begin tran
update Obat set
Obat_Stok = 600 where
Obat_Nama = 'Obat A'
waitfor delay '00:00:20'
commit
```

## Transaksi 2

```
set transaction
isolation level
read committed
select Obat_Stok
from Obat where
Obat_Nama = 'Obat
A'
```



Hasil adalah **600** setelah menunggu delay selama 20 detik karena Query pada Transaksi 2 baru berjalan setelah Query Transaksi 1 di-commit

# READ COMMITTED

**READ COMMITTED** : dapat melihat hanya data-data yang sudah di commit saja, tetapi bisa jadi datanya akan berubah pada waktu yang berbeda


Contoh dalam SQL SERVER dimana Query Transaksi 2 dijalankan setelah Query Transaksi 1

## Transaksi 1

```
begin tran
select * from Obat
waitfor delay '00:00:20'
update      Obat      set
Obat_Stok  =  600  where
Obat_Nama = 'Obat A'
commit
```

## Transaksi 2

```
set      transaction
isolation      level
read committed
select      Obat_Stok
from      Obat      where
Obat_Nama  =  'Obat
A'
```



Hasil adalah **50** (data awal pada database) karena query awal transaksi 1 merupakan select (perintah read table) sehingga tabel Obat tidak di-lock dan Query pada Transaksi 2 dapat berjalan sebelum query Update pada Transaksi 1

# REPEATABLE READ

**REPEATABLE READ** : seperti  
Read Committed dengan  
tambahan data yang di-  
select tidak dapat  
dimodifikasi sampai  
transaksi selesai


Contoh dalam SQL SERVER dimana Query Transaksi 2 dijalankan setelah Query Transaksi 1

## Transaksi 1

```
set transaction isolation level  
repeatable read  
begin tran  
select * from Obat where  
Obat_Nama = 'Obat A'  
waitfor delay '00:00:20'  
select * from Obat where  
Obat_Nama = 'Obat A'  
rollback
```

## Transaksi 2

```
update Obat set  
Obat_Stok = 600  
where Obat_Nama =  
'Obat A'
```



Update pada Transaksi 2 akan berjalan setelah semua transaksi 1 selesai (setelah rollback pada Transaksi 1) karena data Obat A di-lock pada Transaksi 1

# REPEATABLE READ

**REPEATABLE READ** : seperti  
Read Committed dengan  
tambahan data yang di-  
select tidak dapat  
dimodifikasi sampai  
transaksi selesai


*Contoh dalam SQL SERVER dimana Query Transaksi 2 dijalankan setelah Query Transaksi 1*

## Transaksi 1

```
set transaction isolation level  
repeatable read  
begin tran  
select * from Obat  
waitfor delay '00:00:20'  
select * from Obat  
rollback
```

## Transaksi 2

```
Insert into Obat  
values ('B005',  
'Obat E', 300,  
'tablet',25000);
```



Select kedua pada Transaksi 1 akan menampilkan data yang ditambahkan oleh Insert pada Transaksi 2 tanpa *delay* karena repeatable read tidak akan meng-lock untuk perintah insert

# SERIALIZABLE

**SERIALIZABLE** : seperti  
*Repeatable Read* dengan  
tambahan pencegahan  
adanya *Phantom Reads*


*Contoh dalam SQL SERVER dimana Query Transaksi 2 dijalankan setelah Query Transaksi 1*

## Transaksi 1

```
set transaction isolation level  
serializable  
begin tran  
select * from Obat  
waitfor delay '00:00:20'  
select * from Obat  
rollback
```

## Transaksi 2

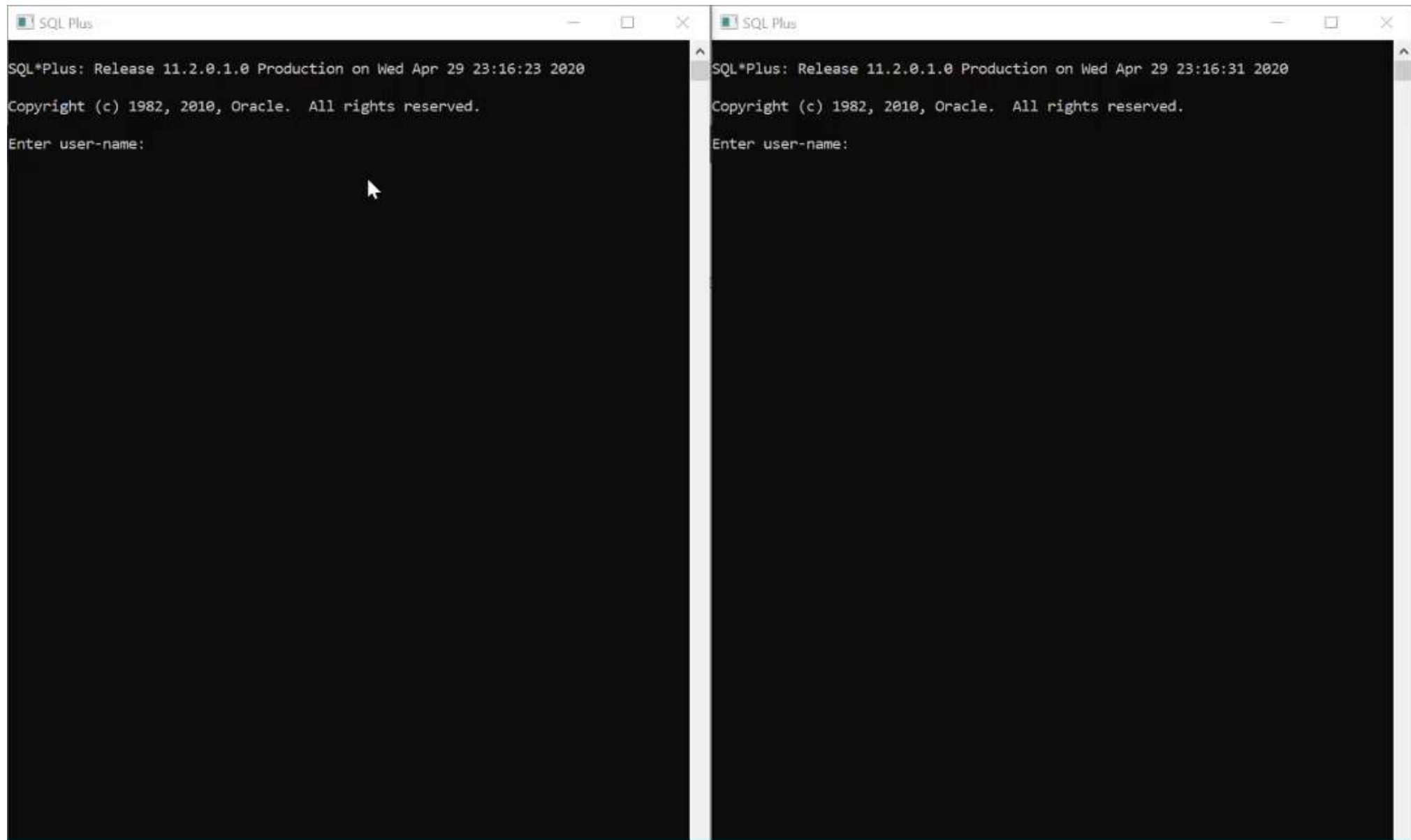
```
Insert into Obat  
values ('B005',  
'Obat E', 300,  
'tablet',25000);
```



Select kedua pada Transaksi 1 akan menghasilkan data yang sama dengan select pertama pada Transaksi 1 karena tabel Obat akan di-lock selama Transaksi 1 berlangsung sehingga insert dilakukan setelah Transaksi 1 selesai

# ISOLATION LEVEL PADA ORACLE

- Secara **DEFAULT**, Isolation Level pada Oracle adalah **READ COMMITTED**
- Dari 4 isolation level yang dijelaskan, Oracle memiliki 2 isolation level yaitu **READ COMMITTED** dan **SERIALIZABLE**
- Di Oracle, **SERIALIZABLE** hanya bisa diimplementasikan pada user selain sysdba user



Sebelah Kiri adalah user system dan Sebelah Kanan adalah user sysdba. User system akan melakukan select dan user sysdba akan melakukan insert



The background of the image is a repeating pattern of green tropical leaves, specifically palm fronds, on a white background. The leaves are scattered across the entire frame, with some appearing more prominently than others. A large, solid light pink rectangular box is centered horizontally and vertically, serving as a backdrop for the text.

**TERIMA KASIH**