Putra Pandu Adikara, S.Kom

Transaction & Conccurency

Basis Data 2

Transaction

Konsep Transaksi

- **Transaction** → suatu unit eksekusi program yang mengakses & mungkin mengupdate berbagai item data
- Misal: transaksi untuk mentransfer $ 50 dari rekening A ke account B:
  - 1. **read** (A)
  - 2. A := A – 50
  - 3. **write** (A)
  - 4. **read** (B)
  - 5. B := B + 50
  - 6. **write** (B)
- Dua isu-isu utama untuk menangani:
  - Kegagalan dari berbagai macam hal, seperti kegagalan hardware dan sistem crash
  - Serentak pelaksanaan beberapa transaksi

Contoh Transfer Uang

- Transaksi untuk mentransfer $ 50 dari rekening A ke account B:
  - 1. **read** (A)
  - 2. A: = A – 50
  - 3. **write** (A)
  - 4. **read** (B)
  - 5. B: = B + 50
  - 6. **write**(B)
- **Persyaratan Atomicity**
  - jika transaksi gagal setelah langkah 3 dan sebelum langkah 6, uang akan "hilang" yang mengarah ke keadaan inkonsistensi database
    - Kegagalan bisa disebabkan oleh perangkat lunak atau perangkat keras
  - sistem harus memastikan bahwa update dari suatu transaksi yang hanya sebagian tidak dilaksanakan dalam database
    - semua atau tidak sama sekali
- **Persyaratan Durability**
  - setelah pengguna telah diberitahu bahwa transaksi tersebut telah selesai (misalnya, transfer dari $ 50 memiliki terjadi), update ke database dengan transaksi harus bertahan bahkan jika ada kegagalan perangkat lunak atau perangkat keras.

Contoh Transfer Uang

- Transaksi untuk mentransfer $ 50 dari rekening A ke account B:
  - 1. **read** (A)
  - 2. A: = A – 50
  - 3. **write** (A)
  - 4. **read** (B)
  - 5. B: = B + 50
  - 6. **write**(B)
- **Persyaratan Consistency** dari contoh di atas
  - jumlah dari A dan B tidak berubah dengan pelaksanaan transaksi
- Secara umum, persyaratan konsistensi termasuk
  - Secara eksplisit integrity constraint ditentukan seperti PK dan FK
  - Implisit integrity constraint
    - misalnya jumlah saldo rekening, jumlah dikurangi jumlah pinjaman harus sama dengan nilai uang tunai di tangan
  - Sebuah transaksi harus melihat database yang konsisten.
  - Selama pelaksanaan transaksi mungkin database untuk sementara tidak konsisten.
  - Setelah transaksi selesai dengan sukses, database harus konsisten
    - Kesalahan logika transaksi dapat mengakibatkan inkonsistensi database

Contoh Transfer Uang

- **Persyaratan Isolation**
  - jika antara langkah 3 dan 6, transaksi T2 lain diijinkan untuk mengakses database yang diperbarui sebagian, maka terjadi inkonsistensi database (jumlah A + B akan kurang dari yang seharusnya).
  - T1 T2

    1. read (A)
    2. A: = A - 50
    3. write (A)
  - read(A), read(B), print(A + B)

    4. read (B)
    5. B := B + 50
    6. menulis (B
- Isolasi dapat dipastikan dengan menjalankan transaksi secara serial
  - yaitu, satu demi satu.
- Namun, menjalankan transaksi serentak memiliki manfaat yang signifikan, seperti yang akan kita lihat nanti.

Properti ACID

- **Transaction** → suatu unit eksekusi program yang mengakses dan mungkin memperbaharui sebagian item data. Untuk menjaga integritas data sistem database harus memastikan ACID:
  - **Atomicity**. Seluruh operasi transaksi tersebut tercermin dalam database atau tidak ada sama sekali.
  - **Consistency**. Pelaksanaan transaksi berada dalam isolasi menjaga konsistensi database.
  - **Isolation**. Meskipun beberapa transaksi dapat mengeksekusi bersamaan, setiap transaksi harus menyadari transaksi konkuren lain yg berjalan. Satu transaksi akan mempengaruhi transaksi lain yang berjalan pada waktu yang sama.
  - **Durability**. Setelah transaksi selesai dengan sukses, perubahan telah dibuat untuk database tetap bertahan, bahkan jika ada kegagalan sistem.

Transaction State

- **Active** → keadaan awal, transaksi tetap di keadaan ini ketika sedang berjalan
- **Partially commited** → setelah pernyataan terakhir telah dieksekusi.
- **Failed** → setelah ada penemuan bahwa eksekusi normal tidak bisa lagi dilanjutkan.
- **Aborted** → setelah transaksi telah di roll back dan database dipulihkan pada keadaan sebelum memulai transaksi.

  Dua pilihan setelah telah dibatalkan:
  - restart transaksi
    - dapat dilakukan hanya jika tidak ada kesalahan logika internal
  - kill transaksi
- **Committed** - setelah berhasil diselesaikan.

Transaction State

partially committed
committed
active
failed
aborted

SQL Transaction

- Untuk melakukan suatu transaksi digunakan syntax:
  - **BEGIN { TRAN |TRANSACTION }** [transaction_name|@tran_name_variable] [**WITH MARK** ['description']]
  - *[database_operation]*
  - **COMMIT { TRAN |TRANSACTION }** [transaction_name|@tran_name_variable]
- Contoh:
  - BEGIN TRAN T1
  - UPDATE Customer SET Name='John' where ID=53
  - COMMIT TRAN T1

- **BEGIN TRANSACTION** mewakili suatu titik keadaan pada saat database konsisten secara logik dan fisik dalam suatu koneksi
- Ketika terjadi kesalahan, semua modifikasi yang dilakukan setelah **BEGIN TRANSACTION** akan dikembalikan ke kondisi konsisten yang diketahui sebelumnya.
- Setiap transaksi berlangsung hingga selesai tanpa error dan **COMMIT TRANSACTION** diberikan untuk membuat modifikasi permanen dalam database, atau error terjadi dan semua modifikasi dihapus melalui pernyataan **ROLLBACK TRANSACTION**

- **WITH MARK** digunakan untuk menandai bahwa transaksi disimpan dalam *transaction log*. Marked transaction dapat digunakan sebagai pengganti waktu ketika me-*restore/recover* database ke kondisi sebelumnya.

```
BEGIN TRANSACTION CandidateDelete
    WITH MARK N'Deleting a Job Candidate';
GO
USE AdventureWorks2008R2;
GO
DELETE FROM AdventureWorks2008R2.HumanResources.JobCandidate
    WHERE JobCandidateID = 13;
GO
COMMIT TRANSACTION CandidateDelete;
GO
```

Nesting Transaction

- Contoh:

```
CREATE TABLE TestTrans(Cola INT PRIMARY KEY,
               Colb CHAR(3) NOT NULL);
GO
CREATE PROCEDURE TransProc @PriKey INT, @CharCol CHAR(3) AS
   BEGIN TRANSACTION InProc
   INSERT INTO TestTrans VALUES (@PriKey, @CharCol)
   INSERT INTO TestTrans VALUES (@PriKey + 1, @CharCol)
   COMMIT TRANSACTION InProc;
GO
/* Start a transaction and execute TransProc. */
BEGIN TRANSACTION OutOfProc;
   GO
   EXEC TransProc 1, 'aaa';
   GO
   /* Roll back the outer transaction, this will roll back TransProc's nested
   transaction. */
ROLLBACK TRANSACTION OutOfProc;
GO
/* The following SELECT statement shows only rows 3 and 4 are still in the table.
This indicates that the commit of the inner transaction from the first EXECUTE
statement of TransProc was overridden by the subsequent rollback. */
SELECT * FROM TestTrans;
GO
```

- Commit dari inner (dalam) transaction diabaikan oleh SQL Server.
- Transaction di-commit atau di-rollback berdasarkan aksi yang diambil pada akhir terluar (outer-most) transaction.
  - `COMMIT TRANSACTION` hanya diaplikasikan pada akhir transaction
- Jika outer transaction di-commit, inner nested transaction juga di-commit, begitu pula jika di-rollback.
- Fungsi **`@@TRANCOUNT`** digunakan untuk mengetahui level nesting transaction.
  - Tiap kali **`COMMIT TRAN`** dilakukan, **`@@TRANCOUNT`** dikurangi 1, jika **`@@TRANCOUNT`**=0 maka sudah tidak lagi di transaction

Transaction Modes

- **Autocommit Transaction**
  - Setiap pernyataan individual adalah transaction (default)
- **Explicit Transaction**
  - Setiap transaction dimulai secara eksplisit dengan pernyataan BEGIN TRANSACTION dan diakhiri secara eksplisit dengan pernyataan COMMIT atau ROLLBACK
- **Implicit Transaction**
  - Setiap transaction secara implisit dimulai setelah pernyataan sebelumnya berakhir, tapi tiap transaction secara eksplisit diselesaikan dengan COMMIT atau ROLLBACK

Kapan menggunakan Implicit?

- Secara default, SQL Server beroperasi dalam mode autocommit, bukan implicit transaction.
- Jika ingin semua perintah membutuhkan **`COMMIT`** atau **`ROLLBACK`** secara eksplisit dalam rangka menyelesaikan transaction, perintah **`SET IMPLICIT_TRANSACTIONS ON`** dapat dijalankan.
- Setiap kali Anda mengeluarkan perintah DML (**`INSERT, UPDATE, DELETE`**), SQL Server secara otomatis melakukan transaksi. Namun, jika Anda menggunakan perintah **`SET IMPLICIT_TRANSACTIONS ON`**, kita dapat mengubahnya sehingga SQL Server menunggu sampai pernyataan eksplisit **`COMMIT ROLLBACK`** dijalankan.
- Hal ini dapat berguna ketika mengeluarkan perintah interaktif, meniru perilaku database lain seperti Oracle.

- Apa yang khas tentang transaksi implisit adalah bahwa penerbitan kembali **SET IMPLICIT_TRANSACTIONS ON** tidak meningkatkan nilai @@TRANCOUNT. COMMIT atau ROLLBACK juga tidak mengurangi nilai @@TRANCOUNT sampai setelah diberikan perintah **SET IMPLICIT_TRANSACTIONS OFF**.
- Developer tidak sering menggunakan transaksi implisit, namun ada pengecualian di ADO. Baca tentang implicit transaction dan ADO
-

Catatan

- Baca dan pahami tentang Transaction di Trigger dan Stored Procedure!

Concurrency

Konsep Concurrency

- Concurrency dapat didefinisikan sebagai kemampuan beberapa proses untuk mengakses atau mengubah data berbagi pakai pada saat yang bersamaan.
  - Semakin besar jumlah proses pengguna bersamaan yang dapat mengeksekusi tanpa menghalangi satu sama lain, semakin besar concurrency dari sistem database.
- Concurrency dipengaruhi oleh
  - suatu proses yang mengubah data mencegah proses-proses lain dari membaca data yang diubah
  - proses yang membaca data mencegah proses-proses lain dari mengubah data tersebut.
- Concurrency juga berdampak ketika beberapa proses berusaha untuk mengubah data yang sama secara bersamaan dan mereka semua sukses tanpa mengorbankan konsistensi data.

Manfaat Concurrency

- Karena beberapa transaksi diperbolehkan untuk berjalan secara bersamaan dalam sistem, keuntungannya adalah:
  - Peningkatan penggunaan (*utilization*) disk dan prosesor, yang menyebabkan *throughput* transaksi yang lebih baik
    - Misalnya satu transaksi dapat menggunakan CPU sementara yang lain membaca dari atau menulis ke disk
  - mengurangi Waktu Respon Rata-Rata (ART) untuk transaksi: transaksi pendek tidak perlu menunggu di belakang transaksi panjang.

Concurrency Control

- Saat melakukan modifikasi tertentu dalam database terkadang dibutuhkan untuk mengunci data sehingga tidak ada orang lain yang dapat melakukan modifikasi data.
- Ada dua pendekatan umum dikenal untuk mengunci database mereka penguncian optimis (*optimistic locking*) dan penguncian pesimis (*pessimistic locking*).
  - Kedua pendekatan ini digunakan untuk mempertahankan concurrency dalam database.

Locking

- **Pessimistic locking,** dilakukan pada baris data source untuk mencegah pengguna memodifikasi data yang akan mempengaruhi pengguna lain.
  - Dalam model pesimis, bila pengguna melakukan tindakan yang menyebabkan *locking* diterapkan, tidak ada orang lain dapat melakukan tindakan sampai di-*release oleh* pemilik yang mengunci.
  - Namun hal ini tidak terjadi dengan model mata optimis.
- **Optimistic locking,** pengguna tidak mengunci baris ketika membaca data, tetapi pengguna hanya mengunci baris ketika melakukan perubahan ke database.

Catatan

- Baca dan pahami tentang:
  - Deadlock
  - Schedule
  - Serializability

Transaction Isolation

Transaction Isolation

- Normalnya sebaiknya menggunakan perlakuan isolation default dari SQL Server untuk transaction. Tapi terkadang kebutuhan bisnis memaksa untuk menggunakan pendekatan isolation yang lain.
- Untuk membantu kasus tsb SQL Server menyediakan 5 model transaction isolation. Namun harus memahami dulu tentang masalah concurrency database:
  - Dirty Reads
  - Non-repeatable Reads
  - Phantom Reads

Masalah Concurrency

- **Dirty Reads →** terjadi ketika transaksi A membaca data yang ditulis transaksi B tapi belum di-commit. Bahayanya dengan 'pembacaan kotor' ini bahwa transaksi B tsb tidak pernah di-commit jadi meninggalkan data 'kotor'
- **Non-repeatable Reads →** terjadi ketika transaksi A mencoba mengakses data yang sama 2 kali dan transaksi B memodifikasi data antara pembacaan transaksi A. Maka akan menyebabkan transaksi A membaca nilai yang berbeda untuk data yang sama, menyebabkan pembacaan asli menjadi non-repeatable
- **Phantom Reads →** terjadi ketika transaksi A mengakses pada rentang sejumlah data lebih dari satu kali dan transaksi B menambahkan atau menghapus data yang berada dalam rentang pada pembacaan transaksi A. Hal ini menyebabkan adanya baris-baris 'phantom' yang tampak atau hilang dari perspektif transaksi A

Transaction Isolation Level

- Model dari isolation pada SQL Server mencoba untuk mengatasi masalah-masalah di atas dengan cara untuk mengimbangi antara kebutuhan bisnis dan transaction isolation.
- Lima level model isolation antara lain:
  - **Read Committed Isolation**
  - **Read Uncommitted Isolation**
  - **Repeatable Read Isolation**
  - **Serializable Isolation**
  - **Snapshot Isolation**

Read Committed Isolation Level

- **Read Committed Isolation →** (default SQL Server) pada model ini database tidak membolehkan transaksi untuk membaca data yang ditulis ke table oleh uncommited transaction. Model ini melindungi dari dirty-reads, tapi tidak untuk melindungi non-repeatable phantom read.
  - (Writers do not wait for Readers)
- Untuk menggunakan Isolation Level ini:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

Read Uncommitted Isolation Level

- **Read Uncommitted Isolation Level →** tidak menyediakan isolation antara transaksi, semua transaksi dapat membaca data yang ditulis oleh uncommitted transaction. Sehingga tidak melindungi dari dirty-reads, phantom reads, dan non-repeatable reads.
- Misal:
- Koneksi pertama membuka transaksi dan mengupdate table Employees

```
USE Northwind
BEGIN TRAN
   -- update the HireDate from 5/1/1992 to 5/2/1992
   UPDATE dbo.Employees SET HireDate = '5/2/1992'
   WHERE EmployeeID = 1
```

- Koneksi kedua membaca data yang sama

```
USE Northwind
SELECT HireDate FROM dbo.Employees
   WHERE EmployeeID = 1
```

Read Uncommitted Isolation Level

- Untuk menggunakan Isolation Level dari koneksi 2

```
USE Northwind

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

SELECT HireDate
FROM dbo.Employees
       WHERE EmployeeID = 1
```

Repeatable Read Isolation Level

- **Repeatable Read Isolation** → satu langkah lebih jauh dari Read Committed model dengan mencegah transaksi dari **menulis (update)** data yang sedang dibaca transaction hingga pembacaan selesai. Isolation ini memproteksi dari dirty-reads dan non-repeatable reads.
- Untuk menggunakan Isolation Level ini:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE
READ
```

Serializable Isolation Level

- **Serializable Isolation →** menggunakan range locks untuk mencegah transaksi dari **insert, update** dan **delete** beberapa baris dalam suatu rentang yang sedang dibaca oleh transaksi lain. Level ini memproteksi dari ketiga masalah concurrency.
  - (Writers wait for Readers)
- Untuk menggunakan Isolation Level ini:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
```

Snapshot Isolation Level

- **Snapshot Isolation Level →** memproteksi dari kesemua masalah concurrency, namun dengan cara yang berbeda. Menyediakan setiap transaksi dengan suatu 'snapshot' dari data yang diminta. Transaksi kemudian mengakses snapshot tersebut untuk semua rujukan di masa depan, mengeliminasi kebutuhan untuk mengembalikan dirty data potensial ke ke table sumber.
- Untuk menggunakan Isolation Level ini:

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
```

Next: Locking

Locking

Resource yang di-Lock

- SQL Server dapat mengunci beberapa resource untuk meningkatkan granularitas

Resource	Description
RID	Row identifier. Used to lock a single row within a table.
Key	Row lock within an index. Used to protect key ranges in serializable transactions.
Page	8 kilobyte –(KB) data page or index page.
Extent	Contiguous group of eight data pages or index pages.
Table	Entire table, including all data and indexes.
DB	Database.

Locking

- Locking adalah bagian utama dari tiap RDBMS karena diperlukan dalam fungsionalitasnya untuk lingkungan multi-user
- Masalah utama dari locking adalah intisarinya pada masalah logical bukan phyisical
- Dalam lingkungan multi-user yang berat masalah logical biasanya muncul cepat atau lambat
- Tiap isolation level menggunakan lock yang berbeda

Lock Modes: Shared Locks (S)

- **Shared Locks (S)**
  - Shared lock digunakan pada data yang sedang dibaca melalui model pessimistic concurrency
  - Digunakan pada operasi yang tidak mengubah data seperti **SELECT**
  - Ketika shared lock digunakan, transaksi lain dapat membaca data tapi tidak bisa mengubah data yang di lock
  - Ketika data yang dilock sudah dibaca, shared lock dilepaskan (released)

Lock Modes: Update Locks (U)

- **Update Locks (U)**
  - Digunakan ketika akan memodifikasi page, kemudian mempromosikan update page lock ke exclusive page lock sebelum melakukan perubahan sebenarnya
  - Digunakan untuk menghindari lock conversion deadlock
  - Hanya satu update lock dapat digunakan pada data satu waktu
  - Mirip dengan exclusive lock, namun tidak bisa memodifikasi data, jadi harus dirubah ke exclusive lock untuk melakukan perubahan

Lock Modes: Exclusive Lock (X)

- **Exclusive Locks (X)**
  - Digunakan untuk operasi modifikasi data, seperti UPDATE, INSERT, DELETE
  - Mengunci data yang sedang dimodifikasi oleh satu transaksi, sehingga mencegah transaksi lain untuk memodifikasi juga.

Lock Modes: Intent Locks (I)

- **Intent Locks (I)**
  - Digunakan oleh transaksi untuk memberitahu transaksi lain bahwa akan berniat melakukan lock terhadap data.
  - Menjamin modifikasi data yang benar/tepat dengan mencegah transaksi lain mendapat lock dari objek yang lebih tinggi dari hirarki lock.

Lock Modes: Schema Locks (S)

- **Schema Locks (S)**
  - **Schema stability lock (Sch-S)**, digunakan ketika membangkitkan execution plan. Lock ini tidak memblok akses pada data objek
  - **Schema modification lock (Sch-M)**, digunakan ketika mengeksekusi pernyataan DML. Memblok akses pada data objek karena strukturnya sedang dirubah

Lock Modes: Bulk Update locks (BU)

- **Bulk Update locks (BU)**
  - Digunakan ketika bulk copying data ke sebuah table
    - Dengan hint menentukan **`TABLOCK`**, atau
    - mengeset table option table **`lock on bulk load`** menggunakan sp_tableoption
  - Lock ini membolehkan proses untuk bulk-copying data secara konkuren ke table yang sama sementara mencegah proses lain yang tidak bulk-copying data untuk mengakses table.

- Spinlocks
  - Merupakan mekanisme penguncian yang ringan yang tidak mengunci data tapi menunggu dalam waktu pendek hingga suatu lock dibebaskan
    - jika sudah ada lock pada data yang mana suatu transaksi juga ingin men-lock data yang sama

Ringkasan Lock Modes

Lock mode	Description
Shared (S)	Digunakan untuk operasi yang tidak mengubah atau memperbarui data (operasi read-only), seperti statemen SELECT.
Update (U)	Mencegah deadlock yang terjadi ketika ada beberapa session untuk membaca, locking, dan update resources nantinya
Exclusive (X)	Digunakan untuk operasi modifikasi data seperti INSERT, UPDATE DELETE Memastikan beberapa update tidak dilakukan pada resource yang sama pada waktu bersamaan
Intent	Digunakan untuk membangun hirarki lock
Schema	Digunakan ketika suatu operasi yang tergantung pada schema dari suatu table sedang dieksekusi
Bulk Update (BU)	Digunakan ketika bulk-copying data ke suatu table dengan menentukan TABLOCK hint

- **Shared lock** kompatibel dengan **Shared lock** atau **Update lock** lainnya.
- **Update lock** kompatibel hanya dengan **Shared lock**
- **Exclusive lock** tidak kompatibel dengan tipe lock lainnya.

Contoh kasus

- Ada 4 proses yang mencoba lock page yang sama pada table yang sama. Proses-proses ini jalan satu-satu setelah lainnya selesai.
  - Proses 1 : SELECT
  - Proses 2 : SELECT
  - Proses 3 : UPDATE
  - Proses 4 : SELECT

Contoh kasus: Penjelasan

- Proses 1 mengeset **Shared lock** pada page, karena tidak ada lock lainnya di page ini
- Proses 2 mengeset **Shared lock** pada page, karena Shared lock kompatibel dengan **Shared lock** lainnya
- Proses 3 ingin memodifikasi data dan ingin mengeset **Exclusive lock**, tapi tidak kompatibel dengan tipe lock lainnya sehingga tidak bisa dilakukan sebelum Proses 1 dan Proses 2 selesai, jadi Proses 3 mengeset **Update Lock**
- Proses 4 tidak bisa mengeset **Shared lock** pada page sebelum Proses 3 selesai, jadi tidak ada **Lock starvation**. Jadi Proses 4 menunggu sampai Proses 3 selesai
  - **Lock starvation** terjadi ketika transaksi read dapat memonopoli table/page, memaksa transaksi write menunggu tidak tentu.
- Setelah Proses 1, 2 selesai, Proses 3 mengubah **Update lock** ke **Exclusive lock** untuk memodifikasi data. Setelah Proses 3 selesai, Proses 4 mengeset **Shared lock** untuk select data

Locking Hint

Locking hint	Description
HOLDLOCK	Hold a shared lock until completion of the transaction in which HOLDLOCK is used. HOLDLOCK is equivalent to SERIALIZABLE.
NOLOCK	Dikenal sebagai dirty reads. Do not issue shared locks and do not honor exclusive locks. Only applies to the SELECT statement.
PAGLOCK	Use page locks where a single table lock would usually be taken.
READCOMMITTED	Perform a scan with the same locking semantics as a transaction running at the READ COMMITTED isolation level. By default, SQL Server 2000 operates at this isolation level.
READPAST	Skip locked rows. This option causes a transaction to skip rows locked by other transactions that would ordinarily appear in the result set, rather than block the transaction waiting for the other transactions to release their locks on these rows. The READPAST lock hint applies only to transactions operating at READ COMMITTED isolation and will read only past row-level locks. Applies only to the SELECT statement.
READUNCOMMITTED	Equivalent to NOLOCK.
REPEATABLEREAD	Perform a scan with the same locking semantics as a transaction running at the REPEATABLE READ isolation level.

Locking Hint

Locking hint	Description
ROWLOCK	Use row-level locks instead of the coarser-grained page- and table-level locks.
SERIALIZABLE	Perform a scan with the same locking semantics as a transaction running at the SERIALIZABLE isolation level. Equivalent to HOLDLOCK.
TABLOCK	Use a table lock instead of the finer-grained row- or page-level locks. SQL Server holds this lock until the end of the statement. However, if you also specify HOLDLOCK, the lock is held until the end of the transaction.
TABLOCKX	Use an exclusive lock on a table. This lock prevents others from reading or updating the table and is held until the end of the statement or transaction.
UPDLOCK	Use update locks instead of shared locks while reading a table, and hold locks until the end of the statement or transaction. UPDLOCK has the advantage of allowing you to read data (without blocking other readers) and update it later with the assurance that the data has not changed since you last read it.
XLOCK	Use an exclusive lock that will be held until the end of the transaction on all data processed by the statement. Can be specified with either PAGLOCK or TABLOCK, in which case the exclusive lock applies to the appropriate level of granularity.

Deadlock

- Deadlock terjadi ketika 2 user mempunyai lock pada objek terpisah dan tiap user ingin me-lock satu sama lain.
- Misal:
  - User 1 me-lock objek A dan ingin me-lock objek B
  - User 2 me-lock objek B dan ingin me-lock objek A
  - Dalam kasus ini SQL Server menghentikan deadlock dengan memilih user sebagai *korban deadlock (deadlock victim)*
  - Kemudian SQL Server me-rollback transaksi user yang gagal, mengirimkan message number 1205 memberitahu aplikasi user adanya kegagalan dan mengijinkan proses user yang tidak gagal untuk tetap lanjut

Deadlock

- Untuk menentukan koneksi mana yang jadi kandidate korban deadlock digunakan **`SET DEADLOCK_PRIORITY`**
- Pada kasus lain SQL Server memilih korban deadlock dengan memilih proses yang menyelesaikan lock circular chain.
- Pada situasi multiuser, aplikasi harus:
  - mengecek error 1205 untuk mengindikasikan transaksi di-rollback
  - Melakukan restart transaksi bila terjadi error 1205

View locks

- Untuk melihat laporan informasi lock-lock yang sedang terjadi gunakan system stored procedure:
  - `sp_lock`
  - `sp_lock2`