PEMROGRAMAN
BASIS DATA

Pertemuan IX

MANAJEMEN TRANSAKSI

Universitas Sanata Dharma
Yogyakarta

TUJUAN

1. Mahasiswa mampu memahami arti dari transaksi
2. Mahasiswa mampu untuk menjelaskan dan memahami tentang proses AUTO COMMIT.
3. Mahasiswa mampu untuk memahami tentang START TRANSACTION, COMMIT dan ROLLBACK
4. Mahasiswa mampu untuk menerapkan konsep transaksi dalam store procedure untuk menginsertkan data.
5. Mahasiswa mampu untuk menerapkan konsep transaksi dalam store procedure untuk menghapus data.

Arti Transaksi

- Transaksi : serangkaian kelompok dari operasi manipulasi database yang dilakukan seolah-olah sebagai satu unit kerja secara individu
- Jika sebuah operasi di dalam transaksi gagal dijalankan, maka keseluruhan transaksi juga akan gagal dijalankan
- 4 properti standar yang dimiliki dari Transaksi pada MySQL : (ACID)
  - Atomicity : memastikan bahwa seluruh operasi dalam unit kerja diselesaikan dengan baik. Jika tidak, transaksi akan dihentikan pada poin kegagalan dan operasi sebelumnya akan dibatalkan sehingga kembali ke keadaan semula.
  - Consistency : memastikan bahwa database secara tepat mengubah keadaan transksi yang berhasil dijalankan dengan commit
  - Isolation : memungkinkan transaksi untuk beroperasi secara independent
  - Durability : memastikan bahwa hasil atau efek dari transaksi dapat bertahan apabila sistem mengalami kegagalan

Auto Commit, Start Transaction, Commit , Rollback

- Fasilitas penanganan kesalahan (error handling) biasa diperlukan untuk mengantisipasi terjadinya kesalahan pada suatu proses transaksi, sehingga programmer bisa mengatur skenario jika suatu operasi gagal sebagian atau seluruhnya.

- Secara default skenario dari transaksi adalah AUTO COMMIT. Artinya semua proses yang berhasil dilaksanakan akan secara otomatis secara fisik disimpan dalam database. Jika diinginkan mulai dari posisi tertentu AUTO COMMIT tidak berfungsi, dapat digunakan perintah START TRANSACTION.

- Selanjutnya sesuatu perintah sesudah pernyataan START TRANSACTION akan ditunda untuk disimpan, sampai bertemu pernyataan COMMIT yang akan menyimpan seluruh proses yang tertunda atau bertemu pernyataan ROLLBACK yang akan membatalkan seluruh proses yang tertunda.

Cont.

Akan tetapi perlu diingat ada beberapa perintah yang tidak dapat di ROLL BACK karena mengandung fungsi COMMIT secara implisit. Perintah tersebut adalah :

- **ALTER TABLE**
- **BEGIN**
- **CREATE INDEX**
- **CREATE TABLE**
- **CREATE DATABASE**
- **DROP DATABASE**
- **DROP INDEX**
- **DROP TABLE**
- **LOAD MASTER DATA**
- **LOCK TABLES**
- **SET AUTOCOMMIT = 1**
- **START TRANSACTION**
- **TRUNCATE TABLE**
- **UNLOCK TABLES**

Syntax

START TRANSACTION | BEGIN [WORK] COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE] ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE] SET AUTOCOMMIT = {0 | 1}

Keterangan:

START TRANSACTION dan BEGIN : digunakan untuk memulai proses transaksi.

COMMIT : untuk mengkomit transaksi tertentu, membuat segala perubahan secara permanen

ROLLBACK : untuk membatalkan tranasksi.

SET AUTOCOMMIT : untuk mengenable dan disable mode default autocommit untuk sebuah koneksi.

Default untuk AUTOCOMMIT adalah enabled

Jenis Tabel MySQL

- Tidak semua jenis tabel MySQL mendukung transaksi.
- Yang mendukung transaksi : INNODB dan BDB

Contoh Mengimplementasikan Transaksi Sederhana

```
create table test.trans(a int, b int) type = INNODB;
begin;
insert into trans(a,b) values(1,2);
select * from trans;
```

	a	b
	1	2

```
Rollback;
Select * from trans;
```

	a	b

Penggunaan FOR UPDATE Untuk Pengganti LOCK Table

- LOCK TABLE tidak dapat digunakan dalam store routine.
- LOCK TABLE hanya dapat digunakan di dalam aplikasi.
- Pernyataan LOCK TABLE dapat diganti dengan pernyataan FOR UPDATE.
- Jadi untuk penguncian baris data (record tertentu) pada tabel, digunakan SELECT .. FOR UPDATE, yang diawali dengan pernyataan START TRANSACTION
- Penguncian dengan FOR UPDATE akan berakhir secara eksplisit setelah transaksi dieksekusi.

Membuat Transaksi Eksplisit

- Sebuah transaksi yang dibentuk dengan menggunakan beberapa transaksi individu menjadi satu kesatuan dan menggunakan perintah eksplisit, yang tujuannya untuk memastikan bahwa seluruh operasi dalam unit kerja diselesaikan dengan baik

Transaksi Implisit

```
DELIMITER $$
DROP PROCEDURE IF EXISTS `pbd`.`spContohTransaksiImplisit`$$
CREATE PROCEDURE `pbd`.`spContohTransaksiImplisit` (pkodespl varchar(20),
    pnofak varchar(30),ptglfak date, pkodebrg varchar(20), phargabeli decimal(10,0),
    pjumlah decimal(10,2))
BEGIN
set max_error_count=0;
set sql_notes=0;set AUTOcommit=0;
start transaction;
insert into beli(kodespl,nofak,tglfak,kodebrg,hargabeli,jumlah) values(pkodespl, pnofak,
    ptglfak, pkodebrg, phargabeli, pjumlah);
        COMMIT;
    update barang set stock=stock+pjumlah  where kodebrg=pkodebrg;
        commit;

END$$DELIMITER ;
```

Transaksi Eksplisit

```
DELIMITER $$
DROP PROCEDURE IF EXISTS `pbd`.`spContohTransaksiEksplisit`$$
CREATE PROCEDURE `pbd`.`spContohTransaksiEksplisit` (pkodespl varchar(20),
      pnofak varchar(30),ptglfak date, pkodebrg varchar(20), phargabeli decimal(10,0),
      pjumlah decimal(10,2))
BEGIN
set max_error_count=0;
set sql_notes=0;set AUTOcommit=0;
start transaction;
insert into beli(kodespl,nofak,tglfak,kodebrg,hargabeli,jumlah) values(pkodespl, pnofak, ptglfak, pkodebrg, phargabeli,
      pjumlah);
set @err=(select @@warning_count);
      if @err=0 then
            COMMIT;
      set @err='';
            else
            ROLLBACK;
      set @err='';
      end if;

update barang set stock=stock+pjumlah  where kodebrg=pkodebrg;
set @err=(select @@warning_count);
            if @err=0 then
                  commit;
                  set @err='';
            else
                  ROLLBACK;
                  set @err='';
            end if;

END$$DELIMITER ;
```

Permasalah *Concurrency* (Pengaksesan Data Secara Bersama)

- Ada 3 permasalah yang berhubungan dengan concurrency:
  - Lost Update Problem
  - *uncommitted dependency problem*
  - *Inconsistent analysis problem*

Pemecahan Permasalah

- Tingkat Isolasi Transaksi :
  - *read-uncommitted,*
  - *read-committed,*
  - *repeatable-read* dan
  - *serializable*
- Jika kita menggunakan lebih dari 1 transaction-safe table, tetapi tidak menggunakan level isolasi SERIALIZABLE (misal repeatable-read) maka ketika satu transaksi diakhiri dengan COMMIT, transaksi lain yang sedang berlangsung daapt melihat perubahan transaksi yang dibuat oleh transaksi pertama. Ini akan membahayakan konsistensi dari sebuah transaksi.
- Jadi, untuk memastikan bahwa seluruh operasi dalam unit kerja diselesaikan dengan baik, jika menggunakan lebih dari 1 transaction-safe, setiap transaksi harus disertai dengan SET TRANSACTION ISOLATION LEVEL REPEATABLE

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
    {READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ |
    SERIALIZABLE}
```

Pemecahan Permasalah (cont)

1. Tingkat isolasi *serializable*
2. *Autocommit* dibuat *disable*
3. Operasi pembacaan data yang akan diikuti dengan operasi *update* data tersebut memberikan *exclusive lock* dengan cara memberikan perintah for *update* setelah perintah *select*.

Tabel Contoh

transactions

```
CREATE TABLE `transactions` (
  `number` int(11) NOT NULL,
  `account` int(11) NOT NULL,
  `date` datetime NOT NULL,
  `balx` int(11) NOT NULL,
  `baly` int(11) NOT NULL,
  `balz` int(11) NOT NULL,
  PRIMARY KEY  (`number`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Lost Update Problem

Time	T_1	T_2	bal_x
t_1		begin_transaction	100
t_2	begin_transaction	read(bal_x)	100
t_3	read(bal_x)	$bal_x = bal_x + 100$	100
t_4	$bal_x = bal_x - 10$	write(bal_x)	200
t_5	write(bal_x)	commit	90
t_6	commit		90

transaksi T1 dieksekusi secara hampir bersamaan dengan transaksi T2, T1 melakukan debet sebesar 10 dari account dengan nilai balx = 100,
sedangkan T2 melakukan kredit sebesar 100 pada account
yang sama. Kedua transaksi ini dijalankan secara bersamaan yang menghasilkan ketidakkonsistenan data dengan nilai akhir account balx = 90.
Jika kedua transaksi ini dieksekusi secara serial (satu setelah yang lain tanpa operasi yang tumpang tindih) akan menjaga konsistensi data dengan nilai akhir account balx = 190 mengabaikan transaksi
yang mana yang dieksekusi lebih dulu.

Contoh SpNonLostUpdateProblem

```
DELIMITER $$
DROP PROCEDURE IF EXISTS `test`.`spNonLostUpdateProblem`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `spNonLostUpdateProblem`()
BEGIN
DECLARE balanceX INTEGER DEFAULT 0;
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;
SET AUTOCOMMIT=0;
SELECT "Waiting ...";
SELECT balx INTO balanceX FROM test.transactions WHERE number=1 FOR
    UPDATE;
SET balanceX = balanceX - 10;
UPDATE test.transactions SET balx=balanceX, date=now() WHERE number=1;
COMMIT;
END$$
DELIMITER ;
```

SpNonLostUpdateProblem2

```
DELIMITER $$
DROP PROCEDURE IF EXISTS `test`.`spNonLostUpdateProblem2`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `spNonLostUpdateProblem2`()
BEGIN
DECLARE balanceX INT;
DECLARE counter INT DEFAULT 0;
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;
SET AUTOCOMMIT=0;
SELECT balx INTO balanceX FROM test.transactions WHERE number=1 FOR UPDATE;
SELECT "Waktu Tunda ...";
WHILE counter < 1000000 DO
SET counter = counter + 1;
END WHILE;
SET balanceX = balanceX + 100;
UPDATE test.transactions SET balx=balanceX, date=now() WHERE number=1;
COMMIT;
END$$
DELIMITER ;
```

Store Procedure dan Transaksi untuk Insert Data

Sebagai contoh akan dibuat sebuah store procedure yang digunakan untuk menginsertkan sebuah transaksi pembelian ke tabel Beli. Pada saat proses transaksi beli terjadi akan mengupdate jumlah barang yang ada di tabel barang. Tetapi sebelum mengupdate stock barang pada tabel barang akan dicek terlebih dahulu apakah barang tersebut ada atau tidak di tabel barang, jika barang yang dibeli tidak ada di dalam tabel barang maka seluruh transaksi akan dibatalkan .

Store Procedure dan Transaksi untuk Insert Data

Berikut struktur tabel barang dan tabel beli yang digunakan:

```
CREATE TABLE `beli` (
          `kodeSpl` varchar(20) default NULL,
          `noFak` varchar(30) default NULL,
          `TglFak` date default NULL,
          `KodeBrg` varchar(20) default NULL,
          `HargaBeli` decimal(10,0) default NULL,
          `Jumlah` decimal(10,0) default NULL
        ) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Store Procedure dan Transaksi untuk Insert Data

```
CREATE TABLE `barang` (
          `kodeBrg` varchar(20) NOT NULL,
          `Nama` varchar(30) default NULL,
          `Satuan` varchar(20) default NULL,
          `HargaBeli` decimal(10,0) default NULL,
          `HargaJual` decimal(10,0) default NULL,
          `Stock` decimal(10,0) default NULL,
          PRIMARY KEY  (`kodeBrg`)
        ) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Store Procedure dan Transaksi untuk Insert Data (cont)

Data pada tabel Barang adalah sebagai berikut :

kodeBrg	Nama	Satuan	HargaBeli	HargaJual	Stock
BG.001	Buku Gambar Dora 10	PCS	900	1000	0
BG.003	Buku Gambar 10 Besar	PCS	1500	1700	0
PS.001	Pensil Standlear 2B	PCS	1200	1500	0
RT.001	Rautan Kecil	PCS	200	250	0
RT.002	Rautan + Setip	PCS	700	850	0
RT.003	Rautan kotak sedang	PCS	(NULL)	(NULL)	(NULL)

Sedangkan data pada tabel Beli masih kosong :

Store Procedure dan Transaksi untuk Insert Data (cont)

Procedur untuk menginsertkan data ke tabel Beli adalah sebagai berikut:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS `pbd`.`spInsertBeli`$$
CREATE PROCEDURE `pbd`.`spInsertBeli`
(pKodeSpl varchar(20),
 pnoFak varchar(30),
 pTglFak varchar(20),
 pKodeBrg varchar(20),
 pHargaBeli numeric,
 pJumlah numeric)
```

Store Procedure dan Transaksi untuk Insert Data (cont)

```
BEGIN
    START TRANSACTION;
    insert into Beli
    values(pKodeSpl,pnoFak,pTglFak,
    pKodeBrg,pHargaBeli,pJumlah);
    if exists(select * from Barang
           where kodeBrg=pKodeBrg) then
       UPDATE Barang set stock
   =if(isnull(stock),0,stock)+pJumlah where kodeBrg =
        pKodeBrg;
       COMMIT;
    else
       ROLLBACK;
    END IF;
END$$
DELIMITER ;
```

Store Procedure dan Transaksi untuk Insert Data (cont)

Dicoba untuk memanggil store procedure spInsertBeli :

- `call spInsertBeli("Aneka","001","2006-10-30","BG.001",900,24);`
- `call spInsertBeli("Aneka","001","2006-10-30","PS.001",1200,60);`
- `call spInsertBeli("Aneka","002","2006-10-31","PS.002",1300,80);`

Pada perintah 1 dan 2 hasilnya akan tercatat ditabel Beli dan mengupdate tabel Barang, tetapi untuk transaksi yang ke -3 tidak tercatat di tabel Beli dan tidak mengupdate di tabel Barang

Store Procedure dan Transaksi untuk Insert Data (cont)

Hasil di tabel Beli :

	kodeSpl	noFak	TglFak	KodeBrg	HargaBeli	Jumlah
	Aneka	001	2006-10-30	BG.001	900	24
	Aneka	001	2006-10-30	PS.001	1200	60
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Store Procedure dan Transaksi untuk Insert Data (cont)

Hasil pada tabel Barang :

	kodeBrg	Nama	Satuan	HargaBeli	HargaJual	Stock
	BG.001	Buku Gambar Dora 10	PCS	900	1000	24
	BG.003	Buku Gambar 10 Besar	PCS	1500	1700	0
	PS.001	Pensil Standlear 2B	PCS	1200	1500	60
	RT.001	Rautan Kecil	PCS	200	250	0
	RT.002	Rautan + Setip	PCS	700	850	0
	RT.003	Rautan kotak sedang	PCS	(NULL)	(NULL)	0
*		(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Store Procedure dan Transaksi untuk Insert Data (cont)

Jika tidak menggunakan perintah START TRANSACTION, ROLLBACK, COMMIT apa yang akan terjadi jika memasukan transaksi BELI tetapi barang yang dibeli belum ada di tabel Barang ???

```
DELIMITER $$
DROP PROCEDURE IF EXISTS `pbd`.`spInsertBeli2`$$
CREATE PROCEDURE `pbd`.`spInsertBeli2`
(pKodeSpl varchar(20),pnoFak varchar(30),
 pTglFak varchar(20),pKodeBrg varchar(20),
 pHargaBeli numeric, pJumlah numeric)
```

Store Procedure dan Transaksi untuk Insert Data (cont)

```
BEGIN
    insert into Beli values(pKodeSpl,pnoFak,pTglFak,
    pKodeBrg,pHargaBeli,pJumlah);
    UPDATE Barang set
   stock=if(isnull(stock),0,stock)+pJumlah
    where kodeBrg=pKodeBrg;
END$$
DELIMITER ;
```

Store Procedure dan Transaksi untuk Hapus Data

Berikut ini akan dicoba untuk membuat store procedure untuk menghapus data. Store procedure akan menghapus data pembelian barang tertentu dari supplier tertentu. Selanjutnya stock yang ada di barang akan disesuaikan. Langkah terakhir adalah jika ternyata data barang pada pembelian sudah pernah diretur maka seluruh perintah akan dibatalkan. Artinya tidak mungkin akan menghapus data pembelian jika barang tersebut sudah pernah diretur.

Store Procedure dan Transaksi untuk Hapus Data (cont)

Struktur tabel untuk RetBeli adalah sebagai berikut:

```
CREATE TABLE `retbeli` (
           `kodeSpl` varchar(20) default NULL,
           `NoRet` varchar(30) default NULL,
           `TglRet` date default NULL,
           `NoFak` varchar(30) default NULL,
           `TglFak` date default NULL,
           `KodeBrg` varchar(20) default NULL,
           `HargaBeli` decimal(10,0) default NULL,
           `Jumlah` decimal(10,0) default NULL
         ) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Store Procedure dan Transaksi untuk Hapus Data (cont)

Tabel awal RetBeli adalah sebagai berikut:

	kodeSpl	NoRet	TglRet	NoFak	TglFak	KodeBrg	HargaBeli	Jumlah
	Aneka	R002	2006-10-31	001	2006-10-30	PS.001	1200	10
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Store Procedure dan Transaksi untuk Hapus Data (cont)

Strore Procedure spHapusBeli adalah sebagai berikut:

```
CREATE PROCEDURE `pbd`.`spHapusBeli` (pKodeSp varchar(20),
pNoFak varchar(30),pTgFak varchar(20),pKodeBrg varchar(20))
BEGIN
  declare vJumlah numeric;
  start transaction;
  select jumlah into vJumlah from Beli where kodeSpl=pKodeSp
  and noFak=pNoFak and TglFak=pTgFak and KodeBrg=pKodeBrg;
  delete from beli where kodeSpl=pKodeSp
  and noFak=pNoFak and TglFak=pTgFak and KodeBrg=pKodeBrg;
  update barang set stock = stock-vJumlah where kodeBrg=pKodeBrg;
  if exists(select * from retbeli where kodeSpl=pKodeSp
  and noFak=pNoFak and TglFak=pTgFak and KodeBrg=pKodeBrg) then
   rollback;
  else
   commit;
  end if;
```

Store Procedure dan Transaksi untuk Hapus Data (cont)

Dicoba untuk memanggil store procedure spHapusBeli sbb:

- `call spHapusBeli("Aneka","001","2006-10-30","BG.001");`
- `call spHapusBeli("Aneka","001","2006-10-30","PS.001");`

Pada perintah 1 akan berhasil menghapus data pada tabel Beli dan mengupdate stock di tabel barang. Sedangkan pada transaksi kedua tidak akan dilakukan proses penghapusan.