artinya kebenaran data dapat diandalkan dan konsistensi data antara tabel yang satu dengan yang lain dapat tetap terjaga.

Lebih jauh, sistem Transaksi pada database memiliki properti ACID (ACIF Property), yaitu: Atomic, Consistent, Durable, dan Isolated. Adapun penjelasannya adalah sebagai berikut:

**Atomic**

Memastikan bahwa semua proses berhasil di eksekusi, jika ada yang gagal, maka semua proses dibatalkan dan semua data yang telah berubah dikembalikan ke kondisi semula, atau lebih simpelnya: semua atau tidak sama sekali

**Consistency**

Data pada database tetap konsisten (sesuai dengan aturan yang ditetapkan) setelah transaksi dilakukan. misal: jika baris suatu tabel gagal untuk mendapatkan ID dari tabel lain maka proses akan dibatalkan karena data pada tabel tersebut terputus, tidak dapat terhubung dengan tabel lain.

**Isolation**

Isolation atau isolasi berarti ketika terjadi transaksi maka data yang sedang diakses, akan di isolir sehingga tidak bisa di akses oleh operasi lain, untuk dapat mengakses data tersebut, harus menunggu operasi pertama selesai.

**Durability**

Ketika transaksi berhasil dilakukan maka perubahan data pada database dilakukan secara permanen, dan tidak akan hilang meskipun terjadi gangguan baik dari sisi software maupun hardware.

## 24.2. Membuat Transaksi

Pada MySQL, transaksi dimulai dengan statemen `START TRANSACTION`, setelah itu kita jalankan berbagai statemen lain untuk memanipulasi data, jika semua statemen tersebut berhasil dijalankan, maka kita jalankan statemen `COMMIT` agar semua perubahan menjadi permanen, atau jika ada

salah satu yang gagal kita batalkan seluruh perubahan dengan perintah `ROLLBACK`

Sebagai contoh, misal kita buat sebuah tabel dengan nama `user` sebagai berikut:

```
CREATE TABLE user (username VARCHAR(50), UNIQUE (username))
ENGINE=InnoDB;
```

Tabel tersebut terdiri dari satu field bernama username yang bersifat unik, yang artinya tidak boleh ada data username yang sama.

Selanjutnya, kita buat transaksi dan tambahkan data pada tabel tersebut. Jalankan query berikut:

```
START TRANSACTION;
INSERT INTO user VALUES('alfa')
INSERT INTO user VALUES('bravo');
```

Sebagai catatan: kita juga dapat memulai transaksi dengan statemen `BEGIN` atau `BEGIN WORK`, namun keduanya bukan standar SQL (tidak digunakan oleh DBMS lain) sehingga disarankan menggunakan `START TRANSACTION`.

Selanjutnya, cek isi dari tabel `user` dengan menjalankan perintah:

```
SELECT * FROM user
```

Hasil yang kita peroleh:

```
+-----------+
| username  |
+-----------+
| alfa      |
| bravo     |
+-----------+
```

Berikutnya, kita jalankan statemen:

```
ROLLBACK;
```

Cek kembali isi dari tabel `user` dengan menjalankan perintah

```
mysql> SELECT * FROM user;
Empty set (0.00 sec)
```

Ternyata sekarang tabel `user` menjadi kosong. Kenapa seperti itu?

Pada MySQL, setiap kali statemen dijalankan, maka MySQL akan melakukan commit yang artinya perubahan akan di simpan secara permanen ke dalam database, misal ketika kita menjalankan statemen `INSERT`, maka ketika MySQL berhasil memasukkan data ke tabel, seketika itu akan melakukan commit yang artinya data tersebut benar benar disimpan pada tabel secara permanen. Tindakan commit seketika tersebut disebut dengan istilah *autocommit*.

Melalui Transaction, kita ubah pola commit tersebut. Dengan statemen `START TRANSACTION`, kita cegah MySQL untuk melakukan commit ketika terjadi perubahan data hingga kita menjalankan statemen `COMMIT` atau `ROLLBACK`.

Pada contoh sebelumnya, ketika kita tambahkan data pada tabel `user`, MySQL belum melakukan commit, meskipun data tersebut telah ditampilkan pada tabel `user`. Selanjutnya, ketika kita jalankan perintah `ROLLBACK`, MySQL membatalkan semua perubahan yang telah terjadi, sehingga tabel `user` menjadi kosong kembali, atau jika kita ubah `ROLLBACK` menjadi `COMMIT`, MySQL akan melakukan commit dengan menyimpan perubahan menjadi permanen, sehingga tabel `user` akan berisi data alfa dan bravo.

Setelah kita menjalankan statemen `COMMIT` atau `ROLLBACK`, maka transaksi berakhir dan pola commit pada MySQL akan kembali seperti semula pada kondisi sebelum statemen `START TRANSACTION` dijalankan. misal jika kita tambahkan data ke tabel `user` maka data tersebut akan langsung disimpan secara permanen.

Contoh lain, perhatikan query berikut ini:

```
mysql> START TRANSACTION;
```

```
Query OK, 0 rows affected (0.08 sec)

mysql> INSERT INTO user VALUES('alfa');
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO user VALUES('alfa ');
ERROR 1062 (23000): Duplicate entry 'alfa' for key 'username'

mysql> ROLLBACK;
Query OK, 0 rows affected (0.04 sec)

mysql> SELECT * FROM user;
Empty set (0.00 sec)
```

Pada contoh diatas, statemen `INSERT` yang pertama berhasil sedangkan yang kedua gagal karena alamat email sama, setelah kita jalankan statemen `ROLLBACK`, semua perubahan akan dibatalkan, termasuk statemen `INSERT` yang pertama, namun jika kita jalankan `COMMIT`, perubahan yang dilakukan oleh statemen `INSERT` yang pertama akan disimpan secara permanen ke tabel `user`.

## 24.3. Membuat Transaksi Dengan Autocommit

Terkait dengan autocommit, secara default autocommit bersifat true, kita dapat membuatnya non aktif dengan menggunakan statemen SET autocommit = 0. Jika autocommit off, maka MySQL tidak akan melakukan commit sampai kita menjalankan statemen `COMMIT` untuk menyimpan perubahan atau `ROLLBACK` untuk mengembalikan data ke kondisi semula.

Perhatikan contoh berikut:

```
mysql> SET autocommit = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO user VALUES('alfa');
Query OK, 1 row affected (0.04 sec)

mysql> SELECT * FROM user;
+----------+
| username |
```

```
+----------+
| alfa     |
+----------+
1 row in set (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT * FROM user;
Empty set (0.00 sec)
```

Pada contoh diatas, meskipun kita telah menambahkan data username alfa ke dalam tabel `user`, namun data tersebut belum disimpan secara permanen. Hal tersebut dikarenakan autocommit tidak aktif, sehingga ketika kita jalankan statemen `ROLLBACK`, data dikembalikan seperti semula, jika kita jalankan statemen `COMMIT`, data tersebut akan disimpan secara permanen. Hal ini mirip dengan transaksi bukan ? (ya, kita bahas di paragraf berikutnya)

Ketika autocommit OFF, maka sejatinya Transaksi sudah berjalan baik ada maupun tidak ada statemen `START TRANSACTION`, selanjutnya, ketika kita menjalankan statemen `COMMIT` atau `ROLLBACK`, maka kita hanya membuat apakan perubahan yang telah terjadi akan disimpan secara permanen atau dibatalkan. Transaksi akan benar benar berhenti sampai kita mengubah autocommit menjadi ON.

Perhatikan contoh berikut:

```
mysql> SET autocommit = 0;
Query OK, 0 rows affected (0.08 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO user VALUES ('alfa');
Query OK, 1 row affected (0.09 sec)

mysql> INSERT INTO user VALUES ('bravo');
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.08 sec)
```

```
mysql> SELECT * FROM user;
+-----------+
| username  |
+-----------+
| alfa      |
| bravo     |
+-----------+
2 rows in set (0.00 sec)
```

Pada query diatas, ketika kita menjalankan statemen `COMMIT`, maka data akan disimpan secara permanen pada database, namun demikian, transaksi belum berakhir, statemen setelahnya akan dianggap MySQL sebagai bagian dari transaksi (meskipun kita tidak menjalankan kembali statemen `START TRANSACTION`), mari kita lanjutkan percobaan diatas:

```
mysql> INSERT INTO user VALUES ('charlie');
Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO user VALUES ('bravo');
ERROR 1062 (23000): Duplicate entry 'bravo ' for key 'username'

mysql> ROLLBACK;
Query OK, 0 rows affected (0.05 sec)

mysql> SELECT * FROM user;
+-----------+
| username  |
+-----------+
| alfa      |
| bravo     |
+-----------+
2 rows in set (0.00 sec)

mysql> SET autocommit = 1;
Query OK, 0 rows affected (0.00 sec)
```

Pada query diatas, setelah autocommit menjadi aktif, maka transaksi berakhir.

Seperti yang telah kita bahas sebelumnya, transaksi akan berakhir ketika kita menjalankan statemen `COMMIT`, `ROLLBACK`, atau mengaktifkan kembali autocommit. Metode tersebut merupakan metode eksplisit untuk menghentikan transaksi, namun demikian transaksi akan otomatis

berhenti dan MySQL akan melakukan commit jika pada transaksi terdapat statemen yang tidak bisa digunakan dalam transaksi, hal ini disebut implisit commit.

Statemen yang tidak dapat digunakan pada transaksi diantaranya statemen yang berkaitan dengan perubahan struktur tabel seperti `CREATE`, `ALTER`, dan `DROP`. List lengkapnya dapat dilihat di http://dev.mysql.com/doc/refman/5.7/en/implicit-commit.html.

Perhatikan contoh berikut:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.12 sec)

mysql> INSERT INTO user VALUES('charlie');
Query OK, 1 row affected (0.05 sec)

mysql> ALTER TABLE user ADD COLUMN `email` VARCHAR(50) NOT NULL
AFTER `username`;
Query OK, 0 rows affected (0.45 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM user;
+----------+
| username |
+----------+
| alfa     |
| bravo    |
| charlie  |
+----------+
3 rows in set (0.00 sec)
```

Pada contoh diatas, sebelum MySQL menjalankan statemen `ALTER TABLE`, MySQL akan melakukan commit, sehingga data username charlie ditambahkan secara permanen ke dalam database dan statemen `ROLLBACK` tidak berpengaruh lagi.

# 24.4. Menggunakan SAVEPOINT

Dengan `SAVEPOINT` kita dapat mengembalikan kondisi transaksi pada titik tertentu. Untuk melakukannya, buat `SAVEPOINT` dengan nama tertentu

pada bagian tertentu transaksi, selanjutnya, untuk mengembalikan transaksi ke kondisi pada SAVEPOINT tersebut, gunakan statemen ROLLBACK TO diikuti nama SAVEPOINT. Perhatikan contoh berikut:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.10 sec)

mysql> INSERT INTO user values('delta');
Query OK, 1 row affected (0.11 sec)

mysql> SAVEPOINT savepoint_1;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO user values('foo');
Query OK, 1 row affected (0.00 sec)

mysql> ROLLBACK TO savepoint_1;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO user values('golf');
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.07 sec)

mysql> SELECT * FROM user;
+----------+
| username |
+----------+
| alfa     |
| bravo    |
| charlie  |
| delta    |
| golf     |
+----------+
5 rows in set (0.00 sec)
```

Pada contoh diatas ketika kita jalankan statemen ROLLBACK TO savepoint_1, maka semua perubahan setelah savepoint_1 akan di batalkan.

# 24.5. Otomasi ROLLBACK

Sejauh ini, kita menjalankan statemen `ROLLBACK` secara manual, artinya kita yang mengidentifikasi kondisi dimana kita akan melakukan rollback. Dalam praktek, kondisi tersebut harus dapat diidentifikasi oleh aplikasi, sehingga `ROLLBACK` dapat dilakukan secara otomatis, misal saat terjadi error pada bagian tertentu.

Agar aplikasi dapat secara otomatis mengidentifikasi kondisi, maka kita perlu ekspresi logika, yang dalam hal ini statemen `IF`, perlu diperhatikan bahwa statemen `IF` ini berbeda dengan fungsi `IF` yang kita gunakan pada statemen `SELECT`, yang telah kita bahas pada BAB terdahulu.

Pada MySQL, statemen `IF` hanya bisa dijalankan pada compound statement yaitu blok statemen yang diawali dengan `BEGIN` dan diakhiri dengan `END`. Compound statement ini hanya ada pada stored routines: `STORED PROCEDURE`, `STORED FUNCTION`, dan `TRIGGER`, sehingga jika transaksi yang kita buat mengandung statemen `IF`, maka harus kita buat di dalam stored routines tersebut.

Misal kita buat transaksi dalam stored procedure bernama `transaksi` sebagai berikut:

```
 1. DELIMITER $
 2. DROP PROCEDURE IF EXISTS transaksi;
 3. CREATE PROCEDURE transaksi ()
 4. BEGIN
 5.       DECLARE sql_error BOOL DEFAULT 0;
 6.
 7.       DECLARE EXIT HANDLER FOR SQLEXCEPTION
 8.       BEGIN
 9.           SET sql_error = 1;
10.       END;
11.
12.       START TRANSACTION;
13.       INSERT INTO user VALUES ('nitro');
14.       INSERT INTO user VALUES ('alfa'); -- Error:
    duplikasi data
15.       INSERT INTO user VALUES ('zebra');
16.
```

```
17.       IF sql_error = 0
18.           THEN COMMIT;
19.       ELSE
20.           ROLLBACK;
21.       END IF;
22. END $
23. DELIMITER ;
```

Jika kita jalankan, maka tidak ada penambahan data pada tabel `user` karena terjadi duplikasi data pada username alfa, jika kita ganti alfa dengan, misal gama, maka data pada tabel `user` akan bertambah 3 username, yaitu: nitro, gama, dan zebra

Pada query diatas, pertama tama kita definisikan variabel sql_error dengan nilai 0, selanjutnya kita gunakan statemen `DECLARE EXIT HANDLER FOR SQLEXCEPTION` untuk menangkap pesan error (ketika terjadi error). Pada statemen tersebut, kita gunakan `DECLARE EXIT` karena ketika terjadi error, maka query akan berhenti (exit), selain `EXIT`, terdapat handler bernama `WARNING` yang akan menangkap jika ada waring). Format penulisan `DECLARE ... HANDLER` adalah:

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
      statemen;
END;
```

Jika statemen hanya satu baris, statemen tersebut bisa langsung ditulis tanpa harus menggunakan `BEGIN...END`, sehingga statemen `DECLARE` pada contoh diatas dapat kita ubah menjadi:

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION SET sql_error = 1
```

Dengan statemen `DECLARE EXIT HANDLER`, maka ketika terjadi error, MySQL akan mengeksekusi statemen tersebut yang akan akan mengubah nilai variabel sql_error menjadi 1, dengan demikian ketika MySQL mengeksekusi statemen `IF`, maka kondisi `ELSE` yang akan terpenuhi, sehingga statemen `ROLLBACK` yang akan dijalankan.

Dalam praktek, cara diatas kurang praktis karena bisa jadi kondisi selalu berubah ubah, yang mengakibatkan syntax pada stored procedure selalu berubah, dan tidaklah praktis jika kita berulang membuat stored procedure, selain itu, kita juga sulit mendeteksi penyebab terjadinya error (kita bahas pada sub BAB 24.6. Membuat Transaksi (Lanjutan)).

Untuk itu, pengujian error dapat dilakukan pada level aplikasi, sehingga kita tidak perlu membuat stored procedure. Contoh berikut implementasi pada PHP:

```
1. <?php
2. $conn = mysqli_connect('localhost', 'root', '', 'tutorial_buku');
3. if (!$conn)
4.         die('Gagal terhubung dengan MySQL');
5.
6. $sql_error = 0;
7.
8. mysqli_begin_transaction($conn);
9. $query = mysqli_query($conn, "INSERT INTO user VALUES ('nitro')");
10. if (!$query)
11.         $sql_error = 1;
12.
13. // ERROR: duplikasi data
14. $query = mysqli_query($conn, "INSERT INTO user VALUES ('alfa')");
15. if (!$query)
16.         $sql_error = 1;
17.
18. $query = mysqli_query($conn, "INSERT INTO user VALUES ('zebra')");
19. if (!$query)
20.         $sql_error = 1;
21.
22. if ($sql_error) {
23.         echo 'Transaksi Gagal';
24.         mysqli_rollback($conn);
25. } else {
26.         echo 'Transaksi Berhasil';
27.         mysqli_commit($conn);
28. }
29.
30. mysqli_close($conn);
```

Pada script diatas, jika query berhasil, maka PHP akan mencetak output ke browser berupa ‘Transaksi Berhasil’ kemudian menjalankan commit, sebaliknya, jika gagal maka PHP akan mencetak output ‘Transaksi Gagal” dan menjalankan rollback.

# 24.6. Membuat Transaksi (Lanjutan)

Setelah kita paham bagaimana transaksi bekerja, selanjutnya mari kita buat transaksi yang melibatkan beberapa tabel. Transaksi ini merupakan implementasi dari proses transaksi penjualan yang kita bahas pada bagian awal bab ini. Pada transaksi ini, kita akan menggunakan tiga tabel yaitu tabel penjualan_detail, tabel penjualan, dan tabel buku. Data yang ada pada ketiga tabel tersebut tampak seperti pada tabel dibawah ini:

Tabel: penjualan

```
+--------+--------------+------------+-----------+
| id_trx | id_pelanggan | tgl_trx    | total_trx |
+--------+--------------+------------+-----------+
|      2 |            3 | 2017-03-22 |    395000 |
|      3 |            2 | 2017-01-10 |    360000 |
|      4 |            1 | 2017-03-04 |    269000 |
|      5 |            4 | 2017-02-15 |    110000 |
|      6 |            3 | 2017-03-07 |    256000 |
|      7 |            2 | 2017-03-05 |    215000 |
|      8 |            3 | 2016-12-12 |    270000 |
|      9 |            3 | 2016-12-12 |    325000 |
+--------+--------------+------------+-----------+
```

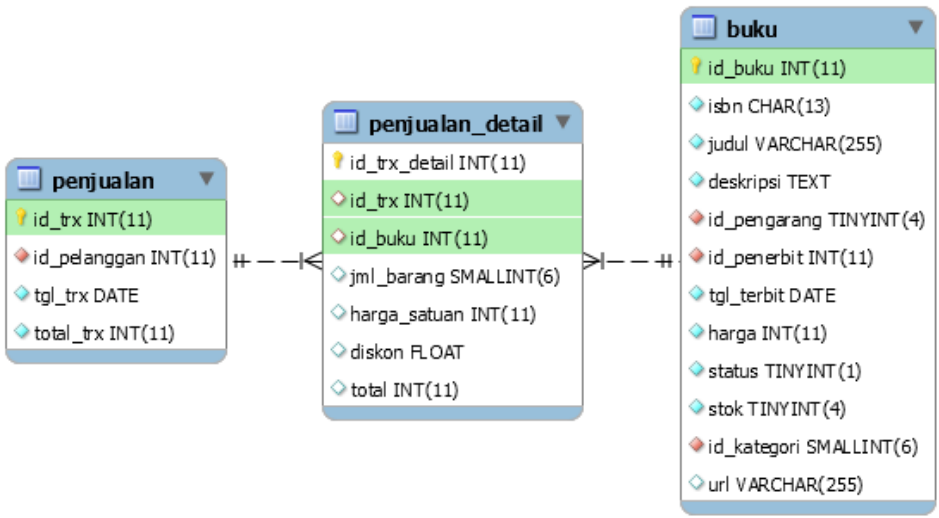
Tabel: penjualan_detail

```
+--------------+--------+---------+------------+--------------+--------+-------+
| id_trx_detail | id_trx | id_buku | jml_barang | harga_satuan | diskon | total |
+--------------+--------+---------+------------+--------------+--------+-------+
|            1 |      2 |       9 |          1 |        46800 |      0 | 46800 |
|            2 |      2 |       4 |          1 |        34800 |      0 | 34800 |
|            3 |      2 |       6 |          2 |        33800 |      0 | 67600 |
|            4 |      3 |       9 |          1 |        46800 |      0 | 46800 |
|            5 |      3 |      10 |          2 |        39800 |      0 | 79600 |
+--------------+--------+---------+------------+--------------+--------+-------+
```

Tabel: buku

```
mysql> SELECT id_buku, judul, harga FROM buku WHERE id_buku IN(3,6,8);
+---------+----------------------+-------+------+
| id_buku | judul                | harga | stok |
+---------+----------------------+-------+------+
|       3 | MySQL Untuk Pemula   | 34800 |    5 |
|       6 | Blogger untuk Pemula | 33800 |    3 |
|       8 | Jago Wordpress       | 39800 |    3 |
+---------+----------------------+-------+------+
3 rows in set (0.00 sec)
```

Hubungan antar tabel seperti tampak pada gambar berikut:




*Gambar 24.1 Relasi antar tabel penjualan, penjualan_detail, dan buku*

Selanjutnya, kita akan memproses data penjualan dengan id_pelanggan 3 dengan rincian sebagai berikut:

- Dua buah buku MySQL Untuk Pemula dengan harga per item Rp. 34.800
- Satu buah buku Blogger Untuk Pemula dengan harga per item Rp. 33.800
- Satu buah buku Jago Wordpress dengan harga per item Rp. 39.800

Langkah yang akan kita lakukan adalah:

- Memasukkan data id_pelanggan dan tgl_trx ke tabel `penjualan` untuk mendapatkan nilai id_trx baru (field id_trx auto increment)
- id_trx tersebut kita gunakan untuk memasukkan data ke tabel `penjualan_detail`.
- Selanjutnya kita hitung total transaksi dan hasilnya kita masukkan ke dalam field nilai_trx yang ada pada tabel `penjualan` yang id_trx nya sama.