DIKLAT TEKNIS UMUM
DESAIN PENGELOLAAN DATABASE

MODUL

Pemrograman SQL

Oleh
Dr. Khamami Herusantoso
Agus Hekso P. S.E., Ak, M.Sc.
Widyaiswara Pusdiklat Keuangan Umum

KEMENTERIAN KEUANGAN REPUBLIK INDONESIA
BADAN PENDIDIKAN DAN PELATIHAN KEUANGAN
PUSDIKLAT KEUANGAN UMUM
JAKARTA
2011

Identitas Modul

Judul Modul:
PEMROGRAMAN SQL

Penulis:
Dr. Khamami Herusantoso
Agus Hekso P. S.E., Ak, M.Sc.

Cetakan Pertama:
2011

Kata Pengantar

Puji syukur kami panjatkan ke hadirat Tuhan yang Maha Esa, karena hanya atas berkat rahmat-Nya kita semua masih diberikan kesempatan untuk melaksanakan tugas-tugas terkait kediklatanhingga saat ini, terutama bagi penulis yang telah diberi kesempatan untuk menyusun dan menyelesaikan modul ini dengan baik.

Modul Pemrograman SQL untuk Diklat Teknis Umum Pengelolaan Database ini disusun oleh Saudara Khamami Herusantoso dan Saudara Agus Hekso Pramudijono berdasarkan Surat Keputusan Kepala Pusdiklat Nomor KEP-003/PP.7/2011 tanggal 28 Januari 2011 tentang Pembentukan Tim Penyusunan Modul Diklat Teknis Umum (DTU) Desain Pengelolaan Database di Lingkungan Pusdiklat Keuangan Umum Tahun Anggaran 2011.

Kami menyetujui modul ini digunakan sebagai bahan ajar bagi para peserta Diklat Teknis Umum Pengelolaan Database. Modul ini merupakan salah satu bahan ajar yang diperlukan selain 4(empat) modul lain yang saling melengkapi yaitu Modul *Pengenalan Konsep Database*, Modul *Desain Database Relasional*, Modul *Structured Query Language*, dan Modul *Pengelolaan Database*, yang kesemuanya menjadi sarana dalam membantu pencapaian tujuan pembelajaran dalam Diklat Teknis Umum Pengelolaan Database.

Akhirnya, semoga Modul Diklatini dapat bermanfaat bagi peserta diklat pada khususnya dan masyarakat luas pada umumnya.

Jakarta, Juni 2011
Kepala Pusat
Pendidikan dan Pelatihan Keuangan Umum

Tony Rooswiyanto
NIP 195604041982031001

Daftar Isi

Daftar Tabel

No table of figures entries found.

Daftar Gambar

No table of figures entries found.

Petunjuk Penggunaan Modul

Modul ini merupakan salah satu bagian dari 5(lima) modul yang diperlukan dan bersifat saling melengkapi, yaitu:

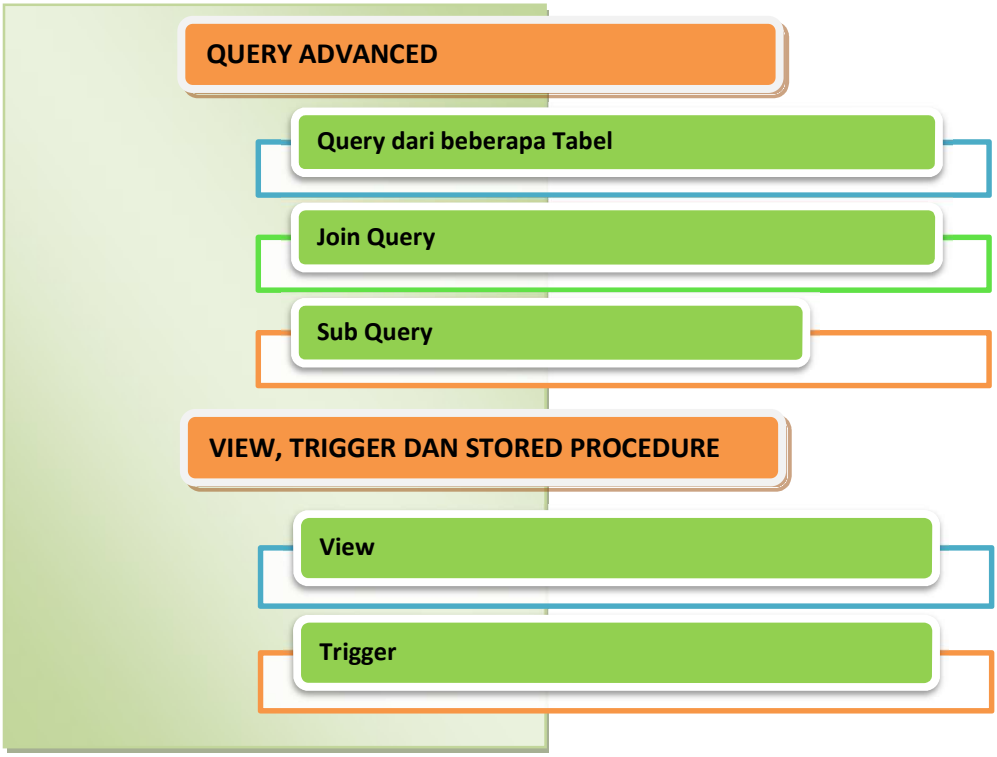
1. Modul Pengenalan Konsep Database;
2. Modul Desain Database Relasional;
3. Modul Structured Query Language;
4. **Modul Pemrograman SQL;**
5. Modul Pengelolaan Database.

yang kesemuanya tersebut menjadi satu paket dan dimaksudkan dalam rangka pencapaian tujuan pembelajaran pada Diklat Teknis Umum Desain Pengelolaan Database.

Modul Pemrograman SQL ini terdiri dariduakegiatan belajar (KB), yaituQuery Advanced sertaView, Trigger dan Stored Procedure, Modul ini perlu untuk dibaca secara berurutan dari KB 1 hingga KB2 agar konsep Pemrograman SQLmenjadi lebih mudah dipahami.

Pada akhir setiap kegiatan belajar diberikan rangkuman yang berisi intisari dari materi yang sudah dibahas sebelumnya.Selanjutnya untuk mengevaluasi pemahaman pembaca, disetiap akhir kegiatan belajar juga disajikan tes formatif. Meskipun sudah disediakan kunci jawaban atas pertanyaan-pertanyaan dalam Tes Formatif, peserta disarankan untuk tidak melihat dulu kunci jawaban, namun sebaiknya peserta mengejakan terlebih dahulu Tes Formatif sesuai dengan alokasi waktu yang diberikanbaru kemudian melakukan penilaian secara mandiri dan mengecek nilainya dengan kriteria umpan balik, apakah sudah tercapai dengan baik. Jika nilai baik belum tercapai, maka peserta disarankan membaca kembali materi dan mengulangi mengerjakan soal tes sampai memperoleh hasil yang diharapkan.

Peta Konsep Modul



A. PENDAHULUAN

1. Deskripsi

Mata pelajaran ini membahas pembuatan Advanced Query meliputi Join dan SubQuery serta pembuatan View, Trigger dan Stored Procedure.

2. Prasyarat Kompetensi

Sebelum mempelajari modul ini, pembaca diharapkan sudah memiliki pengetahuan dasar tentang konsep database dan dasar-dasar SQL.

3. Standar Kompetensi

Setelah mengikuti mata pelajaran ini, peserta diharapkan mampu membuat Advanced Query meliputi Join dan SubQuery serta membuat View, Trigger dan Stored Procedure.

4. Kompetensi Dasar

Setelah selesai mengikuti pembelajaran ini, peserta diklat diharapkan mampu:

a. Membuat Advanced Querydengan benar;
b. Membuat Viewdengan benar;
c. Membuat Triggerdengan benar;
d. Membuat Stored Procedure dengan benar.

5. Relevansi Modul

Setelah mempelajari modul ini diharapkan peserta dapatmengaplikasikannya dalam pekerjaan yang menjadi tugas pokok dan fungsinya.

B. KEGIATAN BELAJAR

1. Kegiatan Belajar 1

Query Advanced

Indikator:
Setelah selesai mengikuti pembelajaran ini peserta diklat diharapkan dapat:

- menjalankan Query dari beberapa Tabel dengan baik;
- menjalankan Join Query dengan baik;
- menjalankan Sub Query dengan baik

a) Query dari beberapa Tabel

Database Relasional adalah suatu model database yang disajikan dalam bentuk tabel.Istilah tabel sering juga disebut dengan relasi atau file. Tabel atau relasi terdiri dari baris dan kolom. Kolom disebut juga dengan atribut atau field. Sedangkan baris disebut juga dengan tupel atau record.

Suatu database relasional terdiri dari kumpulan tabel-tabel. Apabila kita ingin membuat database penjualan produk maka database tersebut akan terdiri dari beberapa tabel misalkan tabel produk, tabel jenis_produk, tabel sales dan tabel sales_item.

Hubungan yang terjadi antara tabel bisa berupa relasi one to many dan many to many. Relasi disebut relasi one to many apabila satu nilai pada sebuah kolom merujuk ke dua atau lebih kolom pada tabel yang lain.Sebagai contoh hubungan antara tabel produk dan jenis_produk adalah hubungan one to many.

```
+----+-------------+
| id | nama        |
+----+-------------+
|  1 | elektronika |
|  2 | furniture   |
|  3 | makanan     |
|  4 | minuman     |
+----+-------------+
4 rows in set (0.00 sec)
```

Tabel jenis_produk

```
mysql> select * from produk;
+------+----------------+---------+------+----------+----------+
| kode | nama           | harga   | stok | min_stok | id_jenis |
+------+----------------+---------+------+----------+----------+
| air7 | air mineral    |    5000 |   40 |       12 |        4 |
| cok2 | coklat         |   15000 |   60 |       12 |        3 |
| ku01 | kulkas 2 pintu | 3000000 |   20 |        4 |        1 |
| lem7 | lemari buku    |  100000 |    9 |        1 |        2 |
| mag6 | magic jar      |  200000 |  200 |       30 |        1 |
| mmk6 | meja makan     |  700000 |   12 |        1 |        2 |
| rot1 | roti manis     |   10000 |  600 |        5 |        3 |
| tv21 | tv lcd 21      | 2000000 |   16 |        4 |        1 |
+------+----------------+---------+------+----------+----------+
8 rows in set (0.00 sec)
```

Tabel produk

Perhatikan bahwa id_jenis pada tabel Produk merujuk pada id pada tabel jenis_produk untuk mengetahui atau memperoleh nama jenis produk (nama). Jadi relasi antar tabeljenis_produk dengan tabel produk adalah relasi one to many.

Berikut perintah SQL untuk membuat table yang mempunyai relasi one to many :

```
mysql> CREATE TABLE jenis_produk (
    -> id       INT(11),
    -> nama  VARCHAR(32) NOT NULL,
    -> PRIMARY KEY(id))
    -> ENGINE=INNODB;
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> CREATE TABLE produk (
    -> kode  VARCHAR(4),
    -> nama  VARCHAR(32),
    -> harga DECIMAL(8),
    -> stok     INT(11),
    -> min_stok INT(11),
    -> id_jenis INT(11),
    -> PRIMARY KEY (kode),
    -> CONSTRAINT fk_id_jenis FOREIGN KEY(id_jenis)
    -> REFERENCES jenis_produk(id) ON DELETE RESTRICT)ENGINE=INNODB;
Query OK, 0 rows affected (0.06 sec)
```

Klausa CONSTRAINT mendefinisikan kolom dalam table yang akan dijadikan FOREIGN KEY yang akan mereferensi (REFERENCES) ke table lain. Definisi FOREIGN KEY sendiri adalahkolom yang menjadi duta / wakil dari kolom lain. Dalam contoh diatas kolom id_jenis pada tableproduk adalah kolom foreign key yang menjadi duta dari table jenis_produk.

Klausa ON DELETE secara spesifik akan melakukan suatu proses jika terjadi penghapusanbaris data pada table jenis_produk. Dalam kasus ini penghapusan tidak diizinkan (DELETE RESTRICT) jika data primary key pada table produk (id) masih digunakan atau direferensi oleh table produk.

Pada bagian ini akan dibahas query yang didasarkan pada beberapa tabel (dua atau lebih tabel) yang saling terkait untuk menghasilkan informasi.

Perintah query SELECT * from produk kurang begitu informatif dari data yang dihasilkannya, karena data jenis_produk hanya ditampilkan id_jenis saja, seharusnya data jenis_produk ditampilkan nama jenis produk. Data nama jenis produk berada di tabel jenis_produk. Oleh sebab itu maka perlu dikaitkan antar tabel produk dengan tabel jenis_produk.

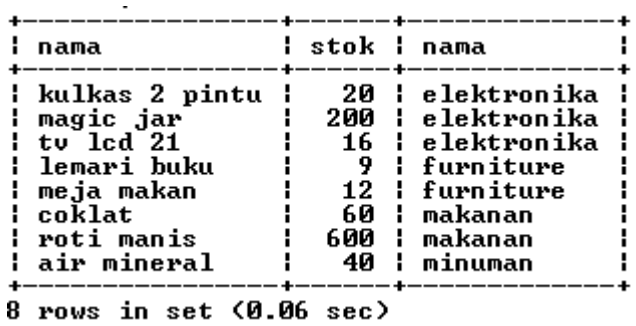
Keterkaitan antara tabel produk dan tabel jenis_produk adalah pada kolom id_jenis di tabel jenis_produk dan kolom id di tabel produk. Penyambungan dilakukan dengan perintah sbb:

produk.id_jenis = jenis_produk.id

Perintah ini akan ditempatkan pada klausa WHERE pada perintah SELECT dan sebelumnya harus disebutkan pada FROM kedua tabel tersebut. Perintah lengkapnya sbb:

```
SELECT produk.nama,produk.stok,jenis_produk.nama
  FROM produk,jenis_produk
WHERE produk.id_jenis = jenis_produk.id
```

Sehingga output yang dihasilkan adalah sbb:



```
+----------------+------+-------------+
| nama           | stok | nama        |
+----------------+------+-------------+
| kulkas 2 pintu |   20 | elektronika |
| magic jar      |  200 | elektronika |
| tv lcd 21      |   16 | elektronika |
| lemari buku    |    9 | furniture   |
| meja makan     |   12 | furniture   |
| coklat         |   60 | makanan     |
| roti manis     |  600 | makanan     |
| air mineral    |   40 | minuman     |
+----------------+------+-------------+
8 rows in set (0.06 sec)
```

Tabel Output Relasi antar Tabel

Bentuk penulisan query diatas dapat juga menggunakan keyword JOIN sebagai pengembangan dari model sebelumnnya. Bentuk penulisan SELECT JOIN adalah sbb:

```
SELECT nama_kolom
  FROM nama_tabel_1
 INNER JOIN nama_tabel_2
    ON kolom_relasi_tabel
```

Berikut contoh penggunaan INNER JOIN dengan menggunakan masalah diatas

```
SELECT produk.nama,produk.stok,jenis_produk.nama
  FROM produk
 INNER JOIN jenis_produk
    ON produk.id_jenis = jenis_produk.id
```

Nama alias dapat diberikan untuk nama kolom dan juga untuk nama tabel. Pemberian nama alias menggunakan keyword AS. Nama alias ini sangat diperlukan untuk menyingkat penulisan-penulisan perintah query yang komplek dan banyak.

Struktur penulisan nama tabel adalah sbb:

```
SELECT nama_kolom
  FROM nama_tabel AS nama_alias
```

Contoh perintah SQL untuk penggunaan alias ini adalah sbb:

```
SELECT P.nama,P.stok,J.nama
```

```
FROM produk AS P, jenis_produk AS J
WHERE P.id_jenis = J.id
```

Perintah INNER JOIN juga dapat menggunakan alias sbb:

```
SELECT P.nama,P.stok,J.nama
 FROM produk AS P
INNER JOIN jenis_produk AS J
  ON P.id_jenis = J.id
```

Pembuatan alias selain menggunakan keyword AS juga dapat menggunakan spasi sebagai pemisah dari nama tabel dan nama alias. Contoh sbb:

```
SELECT P.nama,P.stok,J.nama
 FROM produk P
INNER JOIN jenis_produk J
  ON P.id_jenis = J.id
```

b) Join Query

Perintah *JOIN* ini terdiri dari berbagai variasi, sebelumnya telah dibahas *JOIN* dengan *INNERJOIN*. *INNERJOIN* ini mensyaratkan data di kedua belah tabel sesuai. Data yang tidak sesuai tidak akan ditampilkan sebagai hasil *query* tersebut.

Bentuk variasi lainnya adalah *OUTERJOIN*, perintah *JOIN* ini memungkinkan untuk menampilkan data yang tidak sesuai diantara kedua tabel tersebut ditampilkan. Data yang tidak sesuai nantinya akan diberikan nilai *NULL*. Ketidak sesuaian ini dapat dari satu sisi tabel atau bahkan di kedua sisi tabel tersebut sehingga data *NULL* nya pun dapat muncul di sisi kiri maupun disisi kanan atau bahkan dikedua sisi tersebut.

Penulisan *query* untuk *OUTERJOIN* ini untuk MySQL terbagi menjadi dua yaitu:

1. *LEFTJOIN*
2. *RIGHTJOIN*

Untuk menjalankan perintah *OUTERJOIN* ini akan digunakan tabel sales_item dan tabel produk. Berikut perinta SQL yang digunakan untuk membuat tabel-tabel tersebut:

```
mysql> create table sales_item (
    -> id INT(11),
    -> idsales INT(11),
    -> kode_produk varchar(4),
    -> qty INT(5),
    -> harga DECIMAL(8),
    -> CONSTRAINT fk_kode_produk FOREIGN KEY(kode_produk)
    -> REFERENCES produk(kode) ON DELETE RESTRICT)
    -> ENGINE=INNODB;
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> select * from sales_item;
+------+---------+--------------+------+---------+
| id   | idsales | kode_produk  | qty  | harga   |
+------+---------+--------------+------+---------+
|    1 |       1 | tv21         |    1 | 2000000 |
|    2 |       1 | mag6         |    1 |  200000 |
|    3 |       3 | rot1         |   10 |   10000 |
|    4 |       3 | air7         |   15 |    5000 |
|    5 |       4 | mag6         |    1 |  200000 |
|    6 |       5 | tv21         |    1 | 2000000 |
|    7 |       5 | lem7         |    1 |  100000 |
+------+---------+--------------+------+---------+
7 rows in set (0.00 sec)
```

Bentuk umum penulisan *LEFTJOIN* adalah sbb:

SELECT nama_kolom
  FROM nama_tabel_1
LEFTJOIN nama_tabel_2
    ON relasi_tabel

Perintah *LEFTJOIN* ini akan menghasilkan seluruh baris data pada nama_tabel_1 (yang ada disebelah kiri) walaupun dalam relasi tidak sesuai dengan nama_tabel_2 (disebelah kanan).Ketika data tidak ditemukan pada nama_tabel_2, maka akan diberikan nilai *NULL* pada data tersebut.

Klausa LEFT JOIN digunakan untuk menampilkan seluruh baris data pada table disisi sebelah kiri dari klausa, walaupun ada baris data disisi sebelah kiri yang tidak beririsan atau terhubung dengan kolom yang dihubungkan setelah klausa ON.

Berikut perintah *LEFTJOIN* yang akan menampilkan idsales dari penjualan, kode,nama dari produk dan jumlah quantitas produk yang terjual.

```
mysql> SELECT sales_item.idsales,produk.kode,produk.nama,sales_item.qty
    -> FROM sales_item LEFT JOIN produk
    -> ON sales_item.kode_produk=produk.kode;
+---------+------+-------------+------+
| idsales | kode | nama        | qty  |
+---------+------+-------------+------+
|       1 | tv21 | tv lcd 21   |    1 |
|       1 | mag6 | magic jar   |    1 |
|       3 | rot1 | roti manis  |   10 |
|       3 | air7 | air mineral |   15 |
|       4 | mag6 | magic jar   |    1 |
|       5 | tv21 | tv lcd 21   |    1 |
|       5 | lem7 | lemari buku |    1 |
+---------+------+-------------+------+
7 rows in set (0.04 sec)
```

Penulisan klausa *LEFT JOIN* dapat juga dituliskan *LEFT OUTER JOIN* sehingga perintah SQL nya sbb:

```
mysql> SELECT sales_item.idsales,produk.kode,produk.nama,sales_item.qty
    -> FROM sales_item LEFT OUTER JOIN produk
    -> ON sales_item.kode_produk=produk.kode;
+---------+------+-------------+------+
| idsales | kode | nama        | qty  |
+---------+------+-------------+------+
|       1 | tv21 | tv lcd 21   |    1 |
|       1 | mag6 | magic jar   |    1 |
|       3 | rot1 | roti manis  |   10 |
|       3 | air7 | air mineral |   15 |
|       4 | mag6 | magic jar   |    1 |
|       5 | tv21 | tv lcd 21   |    1 |
|       5 | lem7 | lemari buku |    1 |
+---------+------+-------------+------+
7 rows in set (0.00 sec)
```

Bentuk umum penulisan *RIGHTJOIN* adalah sbb:

SELECT nama_kolom
  FROM nama_tabel_1
RIGHTJOIN nama_tabel_2
    ON relasi_tabel

Perintah *RIGHTJOIN* ini akan menghasilkan seluruh baris data pada nama_tabel_2 (yang ada disebelah kanan) walaupun dalam relasi tidak singkron dengan nama_tabel_1 (disebelah kiri).Ketika data tidak ditemukan pada nama_tabel_1, maka akan diberikan nilai *NULL* pada data tersebut.klausa RIGHT JOIN akan menampilkan seluruh baris data pada table disisi sebelah kanan dari klausa, walaupun ada baris data disisi sebelahkanan yang tidak beririsan/terhubung dengan kolom yang dihubungkan setelah klausa ON

Berikut perintah *RIGHTJOIN* yang akan menampilkan data kode,nama dari produk yang terjual maupun tidak terjual beserta informasi quantitas penjualannya.

```
mysql> SELECT produk.kode,produk.nama,sales_item.qty
    -> FROM sales_item RIGHT JOIN produk
    -> ON sales_item.kode_produk=produk.kode;
+------+----------------+------+
| kode | nama           | qty  |
+------+----------------+------+
| air7 | air mineral    |   15 |
| cok2 | coklat         | NULL |
| ku01 | kulkas 2 pintu | NULL |
| lem7 | lemari buku    |    1 |
| mag6 | magic jar      |    1 |
| mag6 | magic jar      |    1 |
| mmk6 | meja makan     | NULL |
| rot1 | roti manis     |   10 |
| tv21 | tv lcd 21      |    1 |
| tv21 | tv lcd 21      |    1 |
+------+----------------+------+
10 rows in set (0.00 sec)
```

Penulisan klausa *RIGHTJOIN* dapat juga dituliskan *RIGHTOUTERJOIN* sehingga perintah SQL nya sbb:

```
mysql> SELECT produk.kode,produk.nama,sales_item.qty
    -> FROM sales_item RIGHT OUTER JOIN produk
    -> ON sales_item.kode_produk=produk.kode;
+------+----------------+------+
| kode | nama           | qty  |
+------+----------------+------+
| air7 | air mineral    |   15 |
| cok2 | coklat         | NULL |
| ku01 | kulkas 2 pintu | NULL |
| lem7 | lemari buku    |    1 |
| mag6 | magic jar      |    1 |
| mag6 | magic jar      |    1 |
| mmk6 | meja makan     | NULL |
| rot1 | roti manis     |   10 |
| tv21 | tv lcd 21      |    1 |
| tv21 | tv lcd 21      |    1 |
+------+----------------+------+
10 rows in set (0.00 sec)
```

c) Sub Query

Sub*query* adalah sebuah bentuk perintah *SELECT* yang mengembalikan nilai yang ada kepada perintah lain berupa perintah *SELECT*, *INSERT*, *UPDATE* dan *DELETE* atau dengan kata lain sub*query* adalah *query* dalam *query*.

Subquery merupakan alternatif dalam membuat perintah SQL yangmenggunakan *JOIN*, dimana hal ini dibuat untuk meningkatkan *performance* terhadap perintah *query* tersebut. namun demikian, tergantung kepada data jika kalau ingin dibandingkan mana yang lebih baik antara *subquery* dengan *JOIN* ini.

Subquery dapat ditempatkan pada perintah *SELECT* setelah klausa *Select*, *From*, *Where*, *GroupBY* dan *Having*. Umumnya perintah sub *Query* ini mengikuti format sebagai berikut:

- *WHEREekspresi* [*NOT*] *IN*(*subquery*)
- *WHEREekspresi operator perbandingan* [*ANY* | *ALL*] (*subquery*)
- *WHERE* [*NOT*] *EXISTS*(*subquery*)

Untuk melakukan percobaan terhadap perintah sub *Query* ini akan disediakan dua buah tabel yaitu tabel produk dan tabel jenis_produk.

```
mysql> select * from jenis_produk;
+----+-------------+
| id | nama        |
+----+-------------+
|  1 | elektronika |
|  2 | furniture   |
|  3 | makanan     |
|  4 | minuman     |
+----+-------------+
4 rows in set (0.00 sec)

mysql> select * from produk;
+------+----------------+---------+------+----------+-----------+
| kode | nama           | harga   | stok | min_stok | id_jenis  |
+------+----------------+---------+------+----------+-----------+
| air7 | air mineral    |    5000 |   40 |       12 |         4 |
| cok2 | coklat         |   15000 |   60 |       12 |         3 |
| ku01 | kulkas 2 pintu | 3000000 |   20 |        4 |         1 |
| lem7 | lemari buku    |  100000 |    9 |        1 |         2 |
| mag6 | magic jar      |  200000 |  200 |       30 |         1 |
| mmk6 | meja makan     |  700000 |   12 |        1 |         2 |
| rot1 | roti manis     |   10000 |  600 |        5 |         3 |
| tv21 | tv lcd 21      | 2000000 |   16 |        4 |         1 |
+------+----------------+---------+------+----------+-----------+
8 rows in set (0.07 sec)
```

Operator IN

Subquery dengan menggunakan *operatorIN* adalah akan me-*list* hasil dari *subquery* untuk dibandingkan dengan ekspresi *where* yang diberikan. *Subquery* akan dijalankan terlebih dahulu baru kemudian *query* pemanggilnya akan dijalankan.

Berikut contoh perintah *Subquery* untuk menampilkan seluruh produkdengan jenis_produk Elektronika dengan menggunakan *operatorIN*

```
SELECT nama
  FROM produk
WHERE id_jenis IN
(SELECT id
FROM jenis_produk
   WHERE nama='Elektronika')
```

Output yang dihasilkan dari perintah *Query* tersebut adalah:

```
mysql> SELECT nama
    ->   FROM produk
    -> WHERE id_jenis IN
    ->   (SELECT id
    ->      FROM jenis_produk
    ->     WHERE nama='Elektronika')
    -> ;
+----------------+
| nama           |
+----------------+
| kulkas 2 pintu |
| magic jar      |
| tv lcd 21      |
+----------------+
3 rows in set (0.09 sec)
```

Berdasarkan pada perintah *query* diatas, pertama kali akan menjalankan *query* untuk jenis_produk dahulu sehingga akan menghasilkan nilai id = 1. Kemudian akan menjalankan perintah *query* pada produk dengan kondisi id_jenis didasarkan pada hasil *query* pertama. Perintah *query* ini dapat diterjemahkan sbb:

```
SELECT nama
  FROM produk
WHERE id_jenis=1
```

Subquery diatas pun seperti dijelaskan terdahulu bahwa dapat menggunakan perintah *JOIN* dengan bentuk sbb:

```
SELECT DISTINCT P.nama
  FROM produk P
 INNER JOINjenis_produk J
ON P.id_jenis=J.id
   AND J.nama='Elektronika'
```

Jika operator *IN* ditambahkan operator *NOT* maka akan menjadi NOT *IN*. hal ini akan memberikan nilai kebalikan dari hasil yang didapat dengan menggunakan perintah *IN*

```
SELECT nama
  FROM produk
WHERE id_jenisNOT IN
(SELECT id
FROM jenis_produk
   WHERE nama='Elektronika')
```

```
mysql> SELECT nama
    ->   FROM produk
    -> WHERE id_jenis NOT IN
    ->   (SELECT id
    ->      FROM jenis_produk
    ->     WHERE nama='Elektronika')
    -> ;
+-------------+
| nama        |
+-------------+
| air mineral |
| coklat      |
| lemari buku |
| meja makan  |
| roti manis  |
+-------------+
5 rows in set (0.06 sec)
```

Operator *EXISTS*

Operator *Exists* ini akan melakukan pemeriksaan terhadap hasil subqery apakah menghasilkan baris-data atau tidak, jika *subquery* tersebut menghasilkan baris data maka akan mengembalikan nilai true dan sebaliknya jika tidak menghasikan data maka akan mengembalikan nilai false

Berikut contoh perintah Sub*query* untuk menampilkan seluruh produkdengan jenis_produk Elektronikadengan menggunakan operator *EXISTS*

```
SELECT nama
  FROM produk P
WHERE EXISTS
(SELECT *
FROM jenis_produk J
   WHERE P.id_jenis=J.id
  AND J.nama='Elektronika')
```

Output yang dihasilkan dari perintah *Query* tersebut adalah:

```
mysql> SELECT nama
    ->   FROM produk P
    -> WHERE EXISTS
    ->   (SELECT *
    ->      FROM jenis_produk J
    ->     WHERE P.id_jenis=J.id
    ->    AND J.nama='Elektronika')
    -> ;
+----------------+
| nama           |
+----------------+
| kulkas 2 pintu |
| magic jar      |
| tv lcd 21      |
+----------------+
3 rows in set (0.00 sec)
```

Jika operator *EXISTS* ditambahkan operator *NOT* maka akan menjadi *NOTEXISTS*. hal ini akan memberikan nilai kebalikan dari hasil yang didapat dengan menggunakan perintah *EXISTS*

```
SELECT nama
  FROM produk P
WHERE NOT EXISTS
(SELECT *
FROM jenis_produk J
   WHERE P.id_jenis=J.id
  AND J.nama='Elektronika')
```

Output yang dihasilkan dari perintah *Query* tersebut adalah:

```
mysql> SELECT nama
    ->   FROM produk P
    -> WHERE NOT EXISTS
    ->   (SELECT *
    ->      FROM jenis_produk J
    ->     WHERE P.id_jenis=J.id
    ->    AND J.nama='Elektronika')
    -> ;
+-------------+
| nama        |
+-------------+
| air mineral |
| coklat      |
| lemari buku |
| meja makan  |
| roti manis  |
+-------------+
5 rows in set (0.00 sec)
```

Operator Komporasi

Subquery memungkinkan juga menggunakan operator komparasi yang terdiri dari =, <>, >, > =, <, ! >, ! <, or < =. Berbeda dengan operator sebelum penggunaaan operator ini sub*query* harus menghasilkan data tunggal yang menjadi parameter untuk *query* pemanggilnya.

Berikut contoh perintah *Subquery* untuk menampilkan seluruh produkdengan jenis_produk Elektronikadengan menggunakan operator komparasi =

```
SELECT nama
  FROM produk
 WHERE produk.id_jenis =
      (SELECT id
        FROM jenis_produk
       WHERE nama='Elektronika')
```

Output yang dihasilkan dari perintah *Query* tersebut adalah:

```
mysql> SELECT nama
    ->   FROM produk
    ->  WHERE produk.id_jenis =
    ->        (SELECT id
    ->           FROM jenis_produk
    ->          WHERE nama='Elektronika')
    -> ;
+----------------+
| nama           |
+----------------+
| kulkas 2 pintu |
| magic jar      |
| tv lcd 21      |
+----------------+
3 rows in set (0.17 sec)
```

Operator ANY dan ALL

Penggunaan operator ini dipadukan dengan operator relasi sehingga nanti akan terbentuk >*ANY*, =*ANY* , <>*ANY*, >*ALL* dan <>*ALL*

>ANY mengandung arti bahwa akan mengambil nilai lebih besarnya dari nilai terendah yang ada dalam list tersebut. contoh >*ANY* (1,2,3) maka nilai lebih besarnya adalah 1

>*ALL* mengandung arti bahwa akan mengambil nilai lebih besarnya dari nilai tertinggi yang ada dalam lsit tersebut. Contoh >*ALL* (1,2,3) maka nilai lebih besarnya adalah 3

=*ANY* pada dasarnya sama dengan fungsi operator *IN* yaitu akan menyamakan apa yang ada dala list sebagai hasil dari sub*query*nya. <>*ANY* tidak serta merta sama dengan *NOTIN*, hal ini berbeda karena <>*ANY* menghasilkan not = a or not = b , sementara *NOTIN* menghasilkan not = a and not = b. <>*ALL* pengertianya sama dengan *NOTIN*

Berikut contoh perintah Sub*query* untuk menampilkan seluruh produkdengan jenis_produk Elektronika dengan menggunakan operator =*ANY*

```
SELECT nama
  FROM produk
 WHERE produk.id_jenis= ANY
      (SELECT id
         FROM jenis_produk
        WHERE nama='Elektronika')
```

Output yang dihasilkan dari perintah *Query* tersebut adalah:

```
mysql> SELECT nama
    ->   FROM produk
    ->  WHERE produk.id_jenis = ANY
    ->        (SELECT id
    ->           FROM jenis_produk
    ->          WHERE nama='Elektronika')
    -> ;
+----------------+
| nama           |
+----------------+
| kulkas 2 pintu |
| magic jar      |
| tv lcd 21      |
+----------------+
3 rows in set (0.00 sec)
```

d) Latihan

1. Jelaskan perbedaan dari INNER JOIN dan OUTER JOIN dan berikan contohnya
2. Untuk kasus-kasus seperti apakah Left Join dan Right Join ini akan digunakan
3. Apa yang dimaksudkan dengan subquery?
4. Jelaskan karateristik operator Exists yang ada dalam sebuah subquery
5. Jelaskan perbedaan antara operator IN dan ANY dalam sebah subquery

e) Rangkuman

Database relasional terdiri dari tabel-tabel yang saling berhubungan. Untuk mengambil data dari beberapa tabel dapat menggunakan perintah JOIN atau menggunakan subquery.

Subquery adalah sebuah bentuk perintah SELECT yang mengembalikan nilai yang ada kepada perintah lain berupa perintah SELECT, INSERT, UPDATE dan DELETE atau dengan kata lain subquery adalah query dalam query.

f) Tes Formatif Kegiatan Belajar 1

(Waktu: 20 menit)

KUIS BENAR-SALAH

1. Merelasikan antar tabel hanya dapat dilakukan dengan perintah Inner Join saja
2. Pengertian penulisan Left Join sama dengan Left Outer Join
3. Perintah LEFT JOIN ini akan menghasilkan seluruh baris data pada tabel yang ada disebelah kiri walaupun dalam relasi tidak singkron dengan tabel disebelah kanan
4. Penulisan alias tabel hanya dapat dilakukan dengan menambahkan keyword AS
5. Join tabel hanya dapat dilakukan untuk tabel yang dapat dihubungkan salah satu dari kolom data dikedua table
6. Right join adalah sama dengan menampilkan seluruh isi data dari kedua sisi table
7. Operator exists pada sub query hanya memeriksa keberadaan baris data pada sub query tersebut dengan mengembalikan nilai true jika ada dan false jika tidak ada.
8. Operator <>ANY pengertiannya sama dengan NOT IN

UMPAN BALIK DAN TINDAK LANJUT

Periksalah jawaban Saudara dengan kunci jawaban test formatif KB 1. Hitunglah jumlah jawaban Saudara yang benar, kemudian gunakan rumus di bawah ini untuk mengetahui tingkat penguasaan Saudara terhadap materi.

$$\text{Rumus} = \frac{\text{Jumlah jawaban yang benar}}{\text{Jumlah semua soal}} \text{ X } 100\%$$

Penjelasan tingkat penguasaan

0 – 60,99 % = Amat Kurang

61 – 70,99 % = Kurang

71 – 80,99 % = Cukup

81 – 90,99% = Baik

91 – 100% = Amat Baik

Kalau Saudara mencapai tingkat penguasaan 81% atau lebih, maka Saudara dapat meneruskan dengan materi pada KB 2. Tetapi apabila nilai Saudara kurang dari 81%, maka kami sarankan Saudara mengulangi materi pada KB 1, terutama materi yang Saudara belum kuasai.

2. Kegiatan Belajar 2

View, Trigger dan Stored Procedure

Indikator :

Setelah selesai mengikuti pembelajaran ini peserta diklat diharapkan dapat:

- Membuat View dengan benar;
- membuat Trigger dengan benar;
- membuat Stored Procedure dengan benar.

a. View

Views merupakan perintah SELECTyangdisimpan, sehingga setiap saatkita membutuhkannya,kitadapatlangsung memanggilnyatanpaperlumengetikkan perintah SELECTkembali.Views dapatdigunakan untukmempermudahkitadalampembuatanlaporanatau tampilandatabaseyang diinginkandengancepat.

ViewsdiMySQLmulaidisediakanpada versi5.0.Viewsmerupakansuatu tampilantabelvirtual.Views berisiperintahSELECT ketabeldalamdatabase.

Membuat dan Mendefinisikan Views

View dibuat atau didefinisikan dengan menggunakan perintah CREATE VIEW.Bentukumum perintahuntukmembuatview,sebagai berikut:

```
CREATE
    [ORREPLACE]
    [ALGORITHM={UNDEFINED|MERGE|TEMPTABLE}]
    [DEFINER={user|CURRENT_USER}]
    [SQLSECURITY{DEFINER|INVOKER}]
    VIEWview_name[(column_list)]
    ASselect_statement
    [WITH[CASCADED|LOCAL]CHECKOPTION]
```

Berikutinicontohviewuntukmenampilkandata id produk , nama dan jenis produk dari tabel produk.

```
CREATE VIEW `data_produk` AS
(select P.id_jenis,P.nama nama_produk,J.nama jenis_produk
```

```
from produk P,jenis_produk J WHERE
P.id_jenis = J.id)
```

Danuntukmengeksekusiperintahdiatas,kitadapatmemanggildenganperintah SELECTsepertihalnyamenampilkandatadarisuatutabel.Berikutinicontohcara pemanggilan view beserta hasil querynya.

```
SELECT*FROMdata_produk;
```

```
mysql> CREATE VIEW `data_produk` AS
    -> (select P.id_jenis,P.nama nama_produk,J.nama jenis_produk
    -> from produk P,jenis_produk J WHERE
    -> P.id_jenis = J.id)
    -> ;
Query OK, 0 rows affected (0.06 sec)

mysql> select * from data_produk;
+----------+----------------+--------------+
| id_jenis | nama_produk    | jenis_produk |
+----------+----------------+--------------+
|        1 | kulkas 2 pintu | elektronika  |
|        1 | magic jar      | elektronika  |
|        1 | tv lcd 21      | elektronika  |
|        2 | lemari buku    | furniture    |
|        2 | meja makan     | furniture    |
|        3 | coklat         | makanan      |
|        3 | roti manis     | makanan      |
|        4 | air mineral    | minuman      |
+----------+----------------+--------------+
8 rows in set (0.00 sec)
```

Mengubah View

Viewyangsudah dibuat,dapatdiubahdenganperintahALTER.Bentuk umumperintah untukmengubahviewyang sudah ada, sebagai berikut:

```
ALTER
    [ALGORITHM={UNDEFINED|MERGE|TEMPTABLE}]
    [DEFINER={user|CURRENT_USER}]
    [SQLSECURITY{DEFINER|INVOKER}]
    VIEWview_name[(column_list)]
    ASselect_statement
    [WITH[CASCADED|LOCAL]CHECKOPTION]
```

Berikut ini contoh untuk mengubah view yang sudah ada:

```
ALTER `data_produk` AS
(select P.nama nama_produk,J.nama jenis_produk
from produk P,jenis_produk J WHERE
P.id_jenis = J.id)
```

Menghapus View

Viewyangsudahdibuat,dapatdihapusdenganperintahDROP.Berikutini bentuk umum dancontohperintahuntuk menghapus view.

```
DROP VIEW view_name;
```

Contoh:

```
DROP VIEW data_produk;
```

b. Trigger

Trigger digunakan untuk memanggil satu atau beberapa perintah SQL secara otomatis sebelum atau sesudahterjadi proses INSERT,UPDATE atau DELETEdarisuatu tabel.Sebagaicontohmisalnyakitainginmenyimpanid produksecara otomatis ketabel history atau logsebelum menghapusdata di tabel produk.

Adapun manfaat dari trigger antara lain adalah:

- Melakukanupdatedataotomatisjikaterjadiperubahan.Contohnyaadalah dalam sistem penjualan, jika dientri barang baru maka stok akan bertambah secaraotomatis.
- Trigger dapat digunakan untuk mengimplementasikan suatu sistem log. Setiap terjadi perubahan,secara otomatis akanmenyimpanketabel log.
- Trigger dapat digunakan untuk melakukan validasi dan verifikasi data sebelum data tersebut disimpan.

Membuat Trigger Baru

Berikutini bentuk umum perintah untuk membuat triggers:

```
CREATE TRIGGER name
[BEFORE|AFTER] [INSERT|UPDATE|DELETE]
ON tablename
FOR EACH ROW statement
```

dimana

BEFORE|AFTERdigunakanuntukmenentukankapanprosessecaraotomatis akandieksekusi, sebelum atau sesudah proses.

INSERT | UPDATE | DELETE digunakan untuk menentukan event yang dijadikantriggeruntuk menjalankanperintah-perintahdi dalam triggers.

Statement atauperintahdalamtrigger dapatberupasatuperintahsaja,dan dapatjuga beberapaperintahsekaligus.Jikaterdapatbeberapaperintahdalam trigger, maka gunakan perintah **BEGIN** dan **END** untuk mengawali dan mengakhiri perintah.

Didalamstatementtrigger,kitadapatmengaksesrecordtabelsebelum atausesudahprosesdenganmenggunakan**NEW** dan**OLD. NEW** digunakan untukmengambilrecordyangakandiproses(insertatauupdate),sedangkan **OLD**digunakanuntukmengaksesrecord yangsudahdiproses(updateatau delete).

Berikutinicontohtriggeryangakanmencatataktivitasketabel**log**setiap terjadi proses insert ketabel produk:

```
CREATETRIGGERproduk_before_insert
BEFOREINSERTONproduk
FOREACHROW
      INSERTINTOlog(description,datetime,user_id)
      VALUES(New.kode,now(),user());
```

Menghapus Trigger

Untukmenghapustrigger,dapat menggunakanperintah **DROPTRIGGER** dengandiikutidengan namatabeldannamatriggernya.Berikutinibentukumum dancontohperintahuntuk menghapus trigger.

```
DROP TRIGGER tablename.triggername;
```

Contoh:

```
DROP TRIGGER produk_before_insert;
```

c. Stored Procedure

FunctiondanStoredProceduremerupakansuatu kumpulanperintah ataustatementyangdisimpandandieksekusi diserverdatabaseMySQL.Dengan SP(Stored Procedure),kita dapat menyusunprogramsederhanaberbasissintaks SQLuntukmenjalankanfungsitertentu.Halinimenjadikanaplikasiyangkita buat lebih efektifdan efisien.

Berikut ini beberapakeuntungan menggunakan Stored Procedure:

- **Lebihcepat**.Halinikarenakumpulanperintahquerydijalankanlangsungdi server. Berbeda dengan jika dijalankan secara sekuensial di bahasa pemrograman,akanlebihlambatkarenaharus *"bolak-balik"*antaraclientdan server.
- **Menghilangkan duplikasi proses, pemeliharaan yang mudah.** Pada dasarnya operasiyangterjadidi suatuaplikasi terhadapdatabaseadalah sama. Secara umum, di dalam aplikasi biasanya terdapat operasi untuk validasi data inputan, menambahkan record baru,mengubahrecord, menghapus record dan sebagainya. Dengan SP, mungkin kita dapat menghindari adanyaduplikasi prosesyang kurang lebihsama,sehingga pemeliharaannya jugajadi lebih mudah.
- **Meningkatkankeamanandatabase.**DenganadanyaSP,databaseakan lebihamankarenaaplikasiyang memanggilSPtidakperlumengetahuiisidi dalamnya. Sebagaicontoh,dalamproses menambahkandata (insert),kita membuatsuatuSP khusus. Dengandemikian,saatclientatauaplikasiakan menambahkandata (insert)makatidakperlutahunamatabelnya,karena hanya cukupmemanggil SPtersebut denganmengirimkanparameteryang diinginkan.

Selanjutnya,StoredProceduredarisegibentukdansifatnyaterbagimenjadi 2(dua),yaituFUNCTIONdanPROCEDURE. Perbedaanutamaantarafunctiondan procedureadalahterletakpadanilai yang dikembalikannya(di-return).Function memilikisuatunilaiyangdikembalikan(di-return),sedangkanproceduretidak.

Umumnya suatu procedure hanya berisi suatu kumpulan proses yang tidak menghasilnyavalue, biasanya hanya menampilkan saja.

HelloWorld!

Sebagaicontoh sederhana,kitaakanmembuatsuatuSPyangakan menampilkanstring "HelloWorld!"dilayarhasil.Berikutiniperintah query untuk membuat SP tersebut:

```
CREATE PROCEDURE hello()
          SELECT "Hello World!";
```

Untuk memanggil procedure tersebut, gunakanlah CALL. Berikut ini contoh pemanggilan proceduredan hasil tampilannya:

```
CALL hello();
```

Hasilnya sebagai berikut:

```
mysql> CREATE PROCEDURE hello()
    -> SELECT "Hello World!";
Query OK, 0 rows affected (0.24 sec)

mysql> call hello();
+--------------+
| Hello World! |
+--------------+
| Hello World! |
+--------------+
1 row in set (0.06 sec)

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

Membuat,MengubahdanMenghapusSP

Membuat SP

UntukmembuatSPbaru,berikut ini bentuk umumnya:

```
CREATE
    [DEFINER = { user | CURRENT_USER }] PROCEDURE
    sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

CREATE
    [DEFINER = { user | CURRENT_USER }] FUNCTION
    sp_name ([func_parameter[,...]]) RETURNS type
    [characteristic ...] routine_body
```

Contoh1.Procedureuntukmenghitungjumlahstok seluruh produk Elektronik

```
CREATE PROCEDURE jumlahProdukElektronik()
          SELECT SUM(stok) FROM produk
          WHERE id_jenis=1;
```

Carapemanggilandariprocedurediatasadalahdenganmenggunakan **CALL jumlahProdukElektronik()**. Hasilnyaakanditampilkanjumlahstok seluruh produk dari jenis elektronik.

Berikutini bentuk laindari contohdiatas:

```
DELIMITER $$
CREATE PROCEDURE jumlahPelanggan2(OUT hasil AS INT)
     BEGIN
          SELECT COUNT(*) INTO hasil FROM pelanggan;
     END$$
DELIMITER ;
```

Padabentukprocedureyang keduadi atas(**jumlahPelanggan2**), kita menyimpanhasildariprocedurekedalamsatuvariabelbernama**hasil** yang bertipe**INT.**Perbedaandarikeduabentukdiatasadalah,padabentukkedua, kitadapatmemanggilproceduredenganSELECT,sedangkanpadayangpertama tidakbisa. Berikut ini contoh pemanggilan untuk procedure yang kedua:

```
mysql>CALLjumlahPelanggan2(@jumlah);
QueryOK,0rowsaffected(0.00sec)

mysql>SELECT@jumlahAS`JumlahPelanggan`;
+------------------+
|JumlahPelanggan|
+------------------+
|5                 |
+------------------+
1rowinset(0.02sec)
```

Contoh2.Procedureuntukmenghitungjumlahitembarangyangpernah dibeliolehsatupelanggan.

```
DELIMITER$$
CREATEPROCEDURE
    jumlahItemBarang(pelangganVARCHAR(5))
SELECTSUM(detil_pesan.jumlah)
FROMpesan,detil_pesan
WHEREpesan.id_pesan=detil_pesan.id_pesan
ANDpesan.id_pelanggan=pelanggan;
```

Contoh 3. Function untuk menghitungjumlah produk yang tersedia (stock)untuksatuproduktertentu.

```
DELIMITER$$
CREATEFUNCTIONjumlahStockBarang(produkVARCHAR(5))
    RETURNSINT

    BEGIN
        DECLAREjumlahINT;
        SELECTCOUNT(*)INTOjumlahFROMproduk
         WHEREid_produk=produk;
        RETURNjumlah;
    END$$
DELIMITER;
```

Untuk memanggil suatu function, kita tidak menggunakan CALL, tetapi

langsung dapatmemanggil denganSELECT.Berikutini contoh pemanggilanuntuk fungsi di atas.

```
SELECT jumlahStockBarang('B0001');
```

Dan berikut ini hasilnya:

```
+----------------------------+
|jumlahStockBarang('B0001')|
+----------------------------+
|                           1|
+----------------------------+
```

Mengubah SP

Untuk mengubah SP yang sudahada,berikut ini bentuk umumnya:

```
ALTER {PROCEDURE | FUNCTION} sp_name
    [characteristic ...]
```

Menghapus SP

Untukmenghapus SPyang sudah ada,berikut ini bentuk umumnya:

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

SintaksDasardalamSP

SP dapatdikatakan sebagaibahasa pemrogramanyang berada didalam database. Oleh karena itu, tentunya terdapat sintaks-sintaks tertentu berhubungandengan SP tersebut,misalnya bagaimanauntukmendeklarasikan variabel,penyeleksiankondisi,perulangan dsb.Padabagianiniakandiuraikan beberapasintaks dasar SP yang didukung olehMySQL.

Variabel

Variabel digunakan untuk menyimpan suatu nilai secara temporer (sementara)di memory.Variabel akan hilang saat sudah tidak digunakan lagi.

Variabel dalamMySQLsebelumdapatdigunakan, pertamakali harus dideklarasikanterlebihdahulu.Berikutini bentuk umumpendeklarasiansuatu variabel di MySQL:

```
DECLARE variable_name DATATYPE [DEFAULT value];
```

Contohnya:

```
DECLAREjumlahINT;
DECLAREkodeVARCHAR(5);
DECLAREtgl_lahirDATEDEFAULT'1982-10-20';
```

Setelahdideklarasikan,suatuvariabeldapatdiisidengansuatunilaisesuai dengantipedatayangdidefinisikansaatpendeklarasian.Untukmengisikannilaikedalam suatuvariabel,digunakanperintahSET.Formatumumnyasebagai berikut:

```
SET variable_name = expression|value;
```

Contohnya:

```
SETjumlah=10;
SETkode=(SELECTid_pelangganFROMpelangganLIMIT1);
SETtgl_lahir=now();
```

Berikut ini contoh function **hitungUmur()** untuk menghitung umur seseorangsaatini berdasarkantahun kelahiranyangdiberikan.

```
DELIMITER$$
CREATEFUNCTIONhitungUmur(lahirDATE)
    RETURNSINT
    BEGIN
       DECLAREthn_sekarang,thn_lahirINT;
       SETthn_sekarang=YEAR(now());
       SETthn_lahir=YEAR(lahir);
       RETURNthn_sekarang-thn_lahir;
    END$$
DELIMITER;
```

Penyeleksian Kondisi

Dengan adanyafasilitaspenyeleksiankondisi,kitadapatmengaturalur prosesyangterjadi dalamdatabasekita. Di MySQL, penyeleksiankondisi terdiri dari IF,IF...ELSE dan CASE. Berikutini bentuk umum ketigaperintahtersebut:

```
IFkondisiTHEN
    perintah-jika-benar;
ENDIF;

IFkondisiTHEN
    perintah-jika-benar;
ELSE
    perintah-jika-salah;
ENDIF;

CASEexpression
    WHENvalueTHEN
        statements
```

```
[WHENvalueTHEN
    statements...]
[ELSE
    statements]
ENDCASE;
```

Berikutini contohpenggunaanperintahIF dalamfungsi **cekPelanggan**() dimana fungsi ini memeriksa apakah pelanggan sudah pernah melakukan transaksipemesananbarang.Jikasudahpernah,tampilkanpesanberapakali melakukan pemesanan,jika belum tampilkan pesan belum pernahmemesan.

```
DELIMITER $$
CREATE FUNCTION cekPelanggan (pelanggan varchar(5))
    RETURNS VARCHAR (100)
    BEGIN
        DECLARE jumlah INT;
        SELECT COUNT(id_pesan) INTO jumlah FROM pesan
            WHERE id_pelanggan=pelanggan;
        IF (jumlah > 0) THEN
            RETURN CONCAT("Anda sudah bertransaksi sebanyak ",
            jumlah, " kali");
        ELSE
            RETURN "Anda belum pernah melakukan transaksi";
        END IF;
    END$$
DELIMITER ;
```

Dan berikut ini contoh penggunaan perintah CASE dalam fungsi **getDiskon**()dimanafungsiinimenentukandiskonberdasarkanjumlahpesanan yang dilakukan.

```
DELIMITER $$
CREATE FUNCTION getDiskon(jumlah INT) RETURNS int(11)
    BEGIN
        DECLARE diskon INT;
        CASE
            WHEN (jumlah >= 100) THEN
                SET diskon = 10;
            WHEN (jumlah >= 50 AND jumlah < 100) THEN
                SET diskon = 5;
            WHEN (jumlah >= 20 AND jumlah < 50) THEN
                SET diskon = 3;
            ELSE SET diskon = 0;
        END CASE;
        RETURN diskon;
    END$$

DELIMITER ;
```

d. Latihan

1. Apa yang dimaksudkan dengan View?
2. Apa yang dimaksudkan dengan Trigger?
3. Apa yang dimaksudkan dengan Stored Procedure?
4. Apakah View, Trigger dan Stored Procedure dapat saling menggantikan?
5. Sebutkan perbedaan antara function dan stored procedure!

e. RANGKUMAN

View, Trigger dan Stored Procedure merupakan fitur-fitur yang sangat berguna dalam pengelolaan database. Views merupakan perintah SELECT yang disimpan, sehingga setiap saat kita membutuhkannya, kita dapat langsung memanggilnya tanpa perlu mengetikkan perintah SELECT kembali.

Trigger digunakan untuk memanggil satu atau beberapa perintah SQL secara otomatis sebelum atau sesudah terjadi proses INSERT, UPDATE atau DELETE dari suatu tabel.

Stored Procedure merupakan suatu kumpulan perintah atau statement yang disimpan dan dieksekusi di server database MySQL.

TES FORMATIF KEGIATAN BELAJAR 2

(Waktu: 20 menit)

KUIS BENAR-SALAH

1. Trigger digunakan untuk menjamin integritas data, integritas referensial dan proses bisnis
2. Trigger hanya dapat dibuat satu untuk seiap tabel
3. View lebih cepat dibandingkan dengan stored procedure
4. View dapat mengupdate beberapa tabel secara bersamaan
5. Kita dapat membuat stored procedure untuk mengambil informasi dari tabel yang kita tidak mempunyai ijin untuk mengaksesnya
6. Kita dapat menggunakan stored procedure untuk mengambil informasi dari tabel yang kita tidak mempunyai ijin untuk mengaksesnya
7. Ketika kita men drop suatu table maka semua view, trigger dan stored procedure ikut di drop
8. Dengan stored procedure kita dapat menyimpan perintah SELECT dan memanggilnya kembali di waktu yang lain
9. Stored procedure digunakan untuk menyimpan suatu program yang mengakses database di client.
10. View dapat digunakan untuk membatasi akses pengguna terhadap database.

UMPAN BALIK DAN TINDAK LANJUT

Periksalah jawaban Saudara dengan kunci jawaban test formatif KB 2. Hitunglah jumlah jawaban Saudara yang benar, kemudian gunakan rumus di bawah ini untuk mengetahui tingkat penguasaan Saudara terhadap materi.

$$\text{Rumus} = \frac{\text{Jumlah jawaban yang benar}}{\text{Jumlah semua soal}} \times 100\%$$

Penjelasan tingkat penguasaan

0 – 60,99 % = Amat Kurang

61 – 70,99 % = Kurang

71 – 80,99 % = Cukup

81 – 90,99% = Baik

91 – 100% = Amat Baik

Kalau Saudara mencapai tingkat penguasaan 80% atau lebih, maka Saudara dapat memahami modul ini. Tetapi apabila nilai Saudara kurang dari 80%, maka kami sarankan Saudara mengulangi materi pada KB 2, terutama materi yang Saudara belum kuasai.

Penutup

Database relasional terdiri dari tabel-tabel yang saling berhubungan. Untuk mengambil data dari beberapa tabel dapat menggunakan perintah JOIN atau menggunakan subquery.

Beberapa fitur pemrograman database seperti View, Trigger dan Stored Procedure merupakan fitur kunci dalam pembuatan aplikasi database.

Tes Sumatif

Petunjuk: Pilihlah jawaban yang paling tepat!

1. Perhatikan penyataan berikut:
   1. Fungsi agregasi menghasilkan 1 baris data saja
   2. Kondisi filter untuk data pada agregasi tidak menggunakan Where tetapi Having

   A. 1 dan 2 betul
   B. 1 dan 2 salah
   C. 1 betul 2 salah
   D. 1 salah 2 betul
   E. tidak ada jawaban

2. Perhatikan penyataan berikut:
   1. Perintah Join tidak ada dalam struktur penulisan Select
   2. Outer Join akan menampilkan data Null pada data yang tidak ketemu di tabel pembandingnya

   A. 1 dan 2 betul
   B. 1 dan 2 salah
   C. 1 betul 2 salah
   D. 1 salah 2 betul
   E. tidak ada jawaban

3. Perhatikan penyataan berikut:
   1. Fungsi agregasi menghasilkan 1 baris data saja
   2. Kondisi filter untuk data pada agregasi tidak menggunakan Where tetapi Having

   A. 1 dan 2 betul
   B. 1 dan 2 salah
   C. 1 betul 2 salah
   D. 1 salah 2 betul
   E. tidak ada jawaban

4. Pernyataan berikut ini :
   Perintah ini akan menghasilkan seluruh baris data pada tabel yang ada disebelah kiri walaupun dalam relasi tidak sesuai dengan tabel disebelah kanan

   A. JOIN
   B. Left Join
   D. Full Join
   E. tidak ada jawaban

C. Right Join

5. Pernyataan berikut ini :
   Subquery dengan menggunakan operator _____ akan me-list hasil dari subquery untuk dibandingkan dengan ekspresi where yang diberikan. Subquery akan dijalankan terlebih dahulu baru kemudian query pemanggilnya akan dijalankan.
   A. operator IN D. operator any
   B. operator exists E. tidak ada jawaban
   C. operator not exists

6. Jika ada perintah SQL sbb:
   SELECT Nim, Nama, NamaJur
   FROM Mahasiswa, Jurusan
   WHERE _____
   Perintah yang harus diberikan pada Where..
   A. where mahasiswa.kodeJur = jurusan.KodeJur D. kodeJur
   B. kodeJur=KodeJur E. tidak ada jawaban
   C. m.kodeJur = j.KodeJur

Untuk soal no 7:

KODEPENG	NAMAPENG	KOTA
101	HERU	NULL
102	DEWI	JAKARTA
103	HABIB	SEMARANG
104	DESI	NULL

7. Dengan menggunakan tabel diatas perintah subQuery yang dibuat adalah:
   SELECT KodePeng, NamaPeng, Kota
   FROM Pengarang p
   _____ Penerbit t
   ON p.KotaTinggal=t.Kota
   Perintah pada bagian kosong adalah
   A. Inner Join D. Right Join
   B. Left Join E. full Join
   C. Join On

8. Jika ada kondisi operator ANY sbb:

>ANY (1,2,3,4)

Yang dianggap nilai tersebarnya oleh perintah Any ini adalah:

A. 1 D. 4

B. 2 E. >4

C. 3

9. Jika ada kondisi where kode <>ALL maka dapat diganti dengan operator:

   A. Not ANY D. ANY

   B. NOT IN E. ALL

   C. NOT ALL

10. Perintah SQL ini akan menimbulkan error

    SELECT KodeJur, COUNT(Nim) as Jumlah FROM Mahasiswa

    Untuk memperbaikinya maka perlu ditambahkan

    A. WHERE KodeJur D. ORDER BY KodeJur

    B. GROUP KodeJur E. salah semua

    C.GROUP BY KodeJur

11. Perhatikan Tabel berikut EMPLOYEES table: EMP_ID NUMBER(4) NOT NULL LAST_NAME VARCHAR2(30) NOT NULL FIRST_NAME VARCHAR2(30) DEPT_ID NUMBER(2) JOB_CAT VARCHAR2(30) SALARY NUMBER(8,2)

    Pilihlah pernyataan yang menunjukkan department ID, gaji minimum dan gaji maximum yang diberikan di bagian tersebut dengan syarat gaji minimum kurang dari 5000 dan gaji maximum lebih dari 15000?

    A.

```
SELECT dept_id, MIN(salary(, MAX(salary)
FROM employees
WHERE MIN(salary) < 5000 AND MAX(salary) > 15000;
```

    B.

```
SELECT dept_id, MIN(salary), MAX(salary)
FROM employees
WHERE MIN(salary) < 5000 AND MAX(salary) > 15000
GROUP BY dept_id;
```

    C.

```
SELECT dept_id, MIN(salary), MAX(salary)
FROM employees
HAVING MIN(salary) < 5000 AND MAX(salary) > 15000;
```

    D.

```
SELECT dept_id, MIN(salary), MAX(salary)
FROM employees
```

```
GROUP BY dept_id
HAVING MIN(salary) < 5000 AND MAX(salary) < 15000;
```

E.

```
SELECT dept_id, MIN(salary), MAX(salary)
FROM employees
GROUP BY dept_id, salary
HAVING MIN(salary) < 5000 AND MAX(salary) > 15000;
```

Kunci Jawaban

TES FORMATIF KEGIATAN BELAJAR 1

1. b
2. s
3. b
4. s
5. b
6. b
7. s
8. b
9. s
10. b

TES FORMATIF KEGIATAN BELAJAR 2

1. b
2. s
3. b
4. s
5. b
6. b
7. s
8. b
9. s
10. b

TES SUMATIF

1. b
2. c
3. d
4. a
5. d
6. b
7. b
8. b
9. b
10. a
11. c
12. a
13. b
14. c
15. d
16. a
17. b
18. d
19. c
20. a

Daftar Pustaka

1. Codd, E.F (1970), *"A Relational Model of Data for Large Shared Data Banks"*, Communications of The ACM 13(6).
2. Connoly dan Begg (2002), "*Database Systems : A Practical Approach to Design, Implementation and Management"*, 3rd Edition, Pearson Education Limited, USA.
3. Elmasri dan Navathe, "*Fundamentals of Database Systems"*, 3rd Edition, Addison Wesley
4. Abdul Kadir,(2000), Konsep dan Tuntutan Praktis Basis Data, Andi, Yogyakarta
5. Indrajani (2009), *"Sistem Basis Data",* Elex Media Komputindo, Jakarta
6. Ramakrishnan dan Gehrke (2003), "*Database Management Systems*", 3rdEdition, McGraw Hill Companies Inc.