Database Firebird (Bag. 2)

Decky Hendarsyah
dex_3000@yahoo.com

Pada tulisan bagian pertama penulis telah menjelaskan bagaimana instalasi Database Firebird dan pemakaiannya. Untuk tulisan Database Firebird bagian kedua ini penulis ingin mencoba membahas mengenai SQL (Structured Query Language) khususnya untuk Database Firebird beserta contoh kasusnya. SQL adalah bahasa standar untuk melakukan administrasi data di dalam Database. Jadi pada prinsipnya SQL pada Database Firebird dapat dikatakan hampir sama cara pemakaiannya pada Database-Database lainnya yang mendukung SQL seperti MySQL, Oracle, SQLServer dan Database lainnya.

Sebelum membahas SQL sebaiknya kita harus mengetahui beberapa istilah objek yang sering ditemui dalam Database:

- Tabel, adalah menggambarkan dari data yang ditampilkan dalam tabular. Tabel terdiri dari kolom (biasanya disebut field) dan baris (biasanya disebut record).
- View, merupakan tabel virtual yang diambil dari tabel-tabel lain melalui sebuah perintah query (biasanya menggunakan perintah SELECT). View digunakan untuk menyederhanakan query-query yang sangat kompleks dan melibatkan banyak tabel.
- Domain, merupakan objek database yang digunakan untuk mendefinisikan sebuah range tertentu.
- Indeks, merupakan objek database berupa urutan yang didasarkan atas field-field tertentu dalam sebuah tabel.
- Trigger, merupakan objek database yang berupa blok program dan melekat didalam sebuah tabel.
- Stored Procedure, merupakan objek database yang berbentuk modul program dan berisi proses-proses logik tertentu.

Type Data Pada Firebird (Interbase):

- Integer dan smallint
- Float dan double precision
- Numeric and decimal
- Date
- Character
- Blob

KOMPONEN-KOMPONEN SQL:

Dalam SQL terdapat tiga bagian komponen-komponen yaitu sebagai berikut:

A. DDL (Data Defenition Language)
B. DML (Data Manipulation Language)
C. DCL (Data Control Language)

A. DDL (Data Definition Language):

DDL biasanya berhubungan dengan definisi struktur objek dari database seperti perintah CREATE, ALTER, DROP. Fungsi CREATE digunakan untuk pembuatan objek database seperti tabel, domain, view, trigger, stored procedure, maupun objek lainnya. Sedangkan ALTER digunakan untuk mengubah struktur objek database seperti perubahan struktur tabel, perubahan field dalam sebuah view dan lain-lain. Sedangkan DROP digunakan untuk menghapus suatu objek database seperti tabel, field dan lain-lain.

a. Membuat Tabel

Untuk membuat tabel digunakan perintah CREATE dengan perintah umum sebagai berikut:

```
CREATE TABLE namatabel (
   namafield1  tipedata,
   namafield2  tipedata,
   ...
   deklarasi_constraint1,
   deklarasi_constraint2,
   ...
);
```

Contoh:

```
CREATE TABLE barang (
   kodebarang   varchar(10),
   namabarang   varchar(25) NOT NULL,
   kodekategori varchar(6) NOT NULL,
   harga        integer NOT NULL,
   PRIMARY KEY(kodebarang),
   FOREIGN KEY(kodekategori) REFERENCES kategori(kodekategori) ON
   UPDATE CASCADE
);
```

Catatan:

- Untuk field yang dijadikan primary key tidak perlu lagi ditulis NOT NULL karena field sudah otomatis dijadikan NOT NULL.

- Untuk Foreign Key terdapat dua option (dimana kedua option boleh dipakai dan boleh juga tidak):
  a. ON UPDATE:
     Jika data referensi diubah maka semua data yang ada pada tabel yang mereferensi akan ikut berubah secara otomasi.
  b. ON DELETE:
     Jika data referensi dihapus maka semua data yang ada pada tabel yang mereferensi akan ikut terhapus secara otomatis.

b. Membuat Domain:

Domain biasanya digunakan untuk pengganti tipe data pada pembuatan tabel. Bentuk umum perintah membuat domain sebagai berikut:

```
CREATE DOMAIN namadomain AS tipedata CHECK(kondisi);
```

Contoh:

```
CREATE DOMAIN genderdomain AS char(1) CHECK((value='L') or
(value='l') or (value='P') or (value='p'));

CREATE DOMAIN dendadomain AS integer CHECK(value>=0);
```

c. Merubah Struktur Tabel:

Bentuk umum perintah merubah struktur tabel adalah sebagai berikut:

```
ALTER TABLE namatabel
   ADD      field tipedata |
   MODIFY   field tipedatabaru |
   DROP     field;
```

Contoh:

```
- ALTER TABLE anggota ADD alamat varchar(50);
- ALTER TABLE anggota MODIFY nama varchar(30);
- ALTER TABLE anggota DROP alamat;
```

d. Menghapus Tabel

Bentuk umum perintah menghapus tabel adalah sebagai berikut:

```
DROP TABLE namatabel;
```

Contoh:

```
DROP TABLE barang;
```

B. DML (Data Manipulation Language)

DML terdiri atas perintah INSERT, SELECT, UPDATE, DELETE. Perintah INSERT digunakan untuk memasukkan data (record) baru kedalam tabel baik seluruh field maupun sebagian field saja. Perintah SELECT digunakan untuk menampilkan dan menyeleksi data sesuai kondisi yang diinginkan. Sedangkan perintah UPDATE digunakan untuk merubah data dalam tabel. Kemudian perintah DELETE digunakan untuk menghapus data dari tabel.

a. Memasukkan Data:

Bentuk umum perintah memasukkan data kedalam sebuah tabel adalah sebagai berikut:

- Memasukkan seluruh field dalam tabel

```
INSERT INTO namatabel VALUES (nilai1, nilai2, ...);
```

Contoh:

```
INSERT INTO penerbit VALUES ('MZN', 'MIZAN');
```

- Memasukkan sebagian field dalam tabel

```
INSERT INTO namatabel (field1, field2, ...) VALUES
(nilai1, nilai2, ...);
```

Contoh:

```
INSERT INTO anggota (id, nama, jk, telp) VALUES ('1023',
'ANDI', 'PRIA','0274-123456');
```

b. Menyeleksi Data:

Seleksi data (query) merupakan perintah dasar dari SQL yang digunakan untuk menyeleksi data dari satu atau lebih tabel yang terdapat dalam database dengan perintah SELECT.

Bentuk umum perintah seleksi data adalah sebagai berikut:

`SELECT field1, field2, ... FROM namatabel;` (untuk menampilkan beberapa field)

`SELECT * FROM namatabel;` (untuk menampilkan semua field)

Contoh:

```
SELECT id, nama FROM anggota;
SELECT * FROM anggota;
```

c. Merubah Data:

Bentuk umum perintah merubah data adalah sebagai berikut:

```
UPDATE namatabel
SET field1=nilaibaru1, field2=nilaibaru2, ...
WHERE kondisi;
```

Contoh:

```
UPDATE anggota
SET nama='ANDI HENDARTO'
WHERE id='1023';
```

d. Menghapus Data:

Bentuk umum perintah menghapus data adalah sebagai berikut:

`DELETE FROM namatabel;` (menghapus isi tabel)

`DELETE FROM namatabel WHERE kondisi;` (menghapus isi tabel sesuai kondisi)

Contoh:

```
DELETE FROM anggota WHERE id='1023';
```

C. DCL (Data Control Language)

DCL terdiri atas perintah COMMIT, ROLLBACK, GRANT dan REVOKE. Kedua perintah pertama (COMMIT dan ROLLBACK) digunakan untuk mengontrol transaksi yang dilakukan, sedangkan perintah yang kedua (GRANT dan REVOKE) digunakan untuk mengontrol hak akses terhadap user.

a. Mengontrol Transaksi:

COMMIT digunakan untuk menyimpan transaksi secara permanen ke dalam database, sedangkan ROLLBACK merupakan kebalikan dari COMMIT yaitu digunakan untuk menggembalikan data ke dalam keadaan semula (keadaan sebelum transaksi). Data yang telah di COMMIT tidak dapat di ROLLBACK.

Contoh penggunaannya:

```
INSERT INTO penerbit VALUES ('MZN', 'MIZAN');
INSERT INTO penerbit VALUES ('GRM', 'GRAMEDIA');
ROLLBACK;
SELECT * FROM penerbit;
INSERT INTO penerbit VALUES ('ELM', 'ELEX MEDIA');
INSERT INTO penerbit VALUES ('INF', 'INFORMATIKA');
COMMIT;
SELECT * FROM penerbit;
```

b. Mengontrol Hak Akses:

GRANT digunakan untuk memberikan hak akses kepada user, sedangkan statemen REVOKE digunakan untuk mencabut hak akses dari seorang user. Bentuk umumnya adalah sebagai berikut:

```
GRANT HakAkses ON namatabel TO namauser;
REVOKE HakAkses ON namatabel FROM namauser;
```

AGGREGATE FUNCTION

Dalam SQL, terdapat lima buah fungsi aggrgate yang masing-masing digunakan untuk keperluan yang berbeda-beda. Berikut ini fungsi-fungsi yang dimaksud.

Min, yaitu fungsi yang digunakan untuk menentukan nilai minimum dari sebuah field yang terdapat pada sekumpulan record.
Max, yaitu fungsi yang digunakan untuk menentukan nilai maksimum dari sebuah field yang terdapat pada sekumpulan record.
Count, yaitu fungsi yang digunakan untuk menentukan jumlah record yang terdapat pada sekumpulan record.
Sum, yaitu fungsi yang digunakan untuk menghitung jumlah nilai dari sebuah field yang terdapat pada sekumpulan record.
Avg, yaitu fungsi yang digunakan untuk menghitung nilai rata-rata dari sebuah field yang terdapat pada sekumpulan record.

Contoh:

```
SELECT MIN(stok) FROM barang;
SELECT MAX(stok) FROM barang;
SELECT COUNT(stok) FROM barang;
SELECT SUM(stok) FROM barang;
SELECT AVG(stok) FROM barang;
```

PENGURUTAN DATA

Untuk melakukan pengurutan data terhadap hasil query dengan menyertakan perintah ORDER BY dan diikuti dengan nama-nama field yang akan digunakan untuk melakukan pengurutan.

Contoh:

```
SELECT * FROM barang ORDER BY namabarang;
Atau:
SELECT * FROM barang ORDER BY namabarang ASC; (Ascending/Kecil Kebesar)

SELECT * FROM barang ORDER BY namabarang DESC; (Descending/Besar Kekecil)
```

PENGELOMPOKKAN DATA

Data-data hasil proses query dapat dikelompokkan berdasarkan field-field tertentu dengan menggunakan perintah GROUP BY. Pengelompokkan data dilakukan apabila di dalam query yang kita tuliskan terdapat penggunaan fungsi aggredate.

Contoh:

```
SELECT kodebarang, SUM(jumlah) as total FROM penjualan GROUP BY kodebarang;
```

PENYARINGAN DATA

Dalam SQL, penyaringan data dilakukan dengan menggunakan perintah WHERE.

Contoh:

```
• SELECT * FROM barang WHERE stok>15;
• SELECT * FROM barang WHERE (stok>15) AND (namabarang LIKE 'R%');
• SELECT * FROM barang WHERE (stok>15) AND (namabarang LIKE '%DENT%');
• SELECT kodebarang, SUM(jumlah) as total FROM penjualan GROUP BY kodebarang HAVING sum(jumlah)>4;
```

JOIN

Operasi join digunakan untuk menggabungkan dua buah tabel yang direlasikan berdasarkan field tertentu di dalam query.

a. Cross Product

Mula-mula operasi join dari duah buah tabel akan memberikan hasil yang merupakan cross product (sering disebut juga dengan cartesian product). Operasi join semacam ini juga dinamakan dengan operasi cross join.

Contoh:

```
SELECT * FROM T1, T2;
Atau:
SELECT * FROM T1 cross join T2;
```

b. Inner Join

Inner Join adalah menggabungkan dua buah tabel untuk menampilkan data-data yang field relasinya bernilai sama.

Contoh:

```
SELECT * FROM T1, T2 WHERE T1.A=T2.A;
Atau:
SELECT * FROM T1 INNER JOIN T2 ON T1.A=T2.A;
```

c. Outer Join

Outer Join adalah proses penggabungan dua buah tabel dimana data-data yang tidak sesuai (tidak memiliki pasangan) antara kedua tabel tersebut juga akan ikut ditampilkan.

- **Left Outer Join**

  Akan menampilkan semua data yang terdapat pada tabel pertama (bagian kiri) meskipun data tersebut tidak memiliki pasangan dengan data yang terdapat pada tabel kedua (bagian kanan).

  Contoh:

```
SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.A=T2.A;
```

  Atau:

```
SELECT * FROM T1 LEFT JOIN T2 ON T1.A=T2.A;
```

- **Right Outer Join**

  Merupakan kebalikan dari left outer join. Operasi ini akan menampilkan semua data yang terdapat pada tabel kedua (bagian kanan) meskipun data tersebut tidak memiliki pasangan dengan data yang terdapat pada tabel pertama (bagian kiri).

  Contoh:

```
SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.A=T2.A;
```

  Atau:

```
SELECT * FROM T1 RIGHT JOIN T2 ON T1.A=T2.A;
```

- **Full Outer Join**

  Merupakan gabungan dari left outer join dan right outer join. Akan menampilkan semua data yang terdapat pada kedua buah tabel (bagian kiri dan kanan) baik yang memiliki pasangan maupun tidak.

  Contoh:

```
SELECT * FROM T1 FULL OUTER JOIN T2 ON T1.A=T2.A;
```

KASUS 1: INVENTORI BARANG

1. Tambahkan seorang *user* pada server Firebird dengan nama dan *password* dexkai serta lengkapi informasi nama depan dan nama belakang untuk *user* tersebut!

   Jawab:

```
- gsec -user SYSDBA -password masterkey
- add dexkai -pw dexkai -fname Decky lname Hendarsyah
```

2. Buat *database* pada server Firebird dengan nama barang (boleh menggunakan *user* SYSDBA ataupun *user* lain)!

   Jawab:

```
- isql -user SYSDBA -password masterkey
- CREATE DATABASE 'localhost:d:\FDB\barang.fdb';
```

3. Buat tabel-tabel di bawah ini di dalam *database* soal nomor 2!
   Jawab:

```
- CREATE DOMAIN jumlahbarangdomain AS Integer
  Check(Value>0);
- CREATE TABLE kategori (kodekategori varchar(6) NOT NULL,
  namakategori varchar(20) NOT NULL, PRIMARY
  KEY(kodekategori));
- CREATE TABLE pemasok (kodepemasok varchar(6) NOT NULL,
  namapemasok varchar(25) NOT NULL, alamatpemasok
  varchar(40) NOT NULL, PRIMARY KEY(kodepemasok));
- CREATE TABLE barang (kodebarang varchar(6) NOT NULL,
  namabarang varchar(20) NOT NULL, kodekategori varchar(6)
  NOT NULL, PRIMARY KEY(kodebarang), FOREIGN
  KEY(kodekategori) REFERENCES kategori(kodekategori) ON
  UPDATE CASCADE);
- CREATE TABLE masuk (nomormasuk INT NOT NULL, tanggalmasuk
  DATE NOT NULL, kodebarang varchar(6) NOT NULL, kodepemasok
  varchar(6), jumlahbarang jumlahbarangdomain NOT NULL,
  totalharga INT NOT NULL, PRIMARY
  KEY(nomormasuk,tanggalmasuk), FOREIGN KEY(kodebarang)
  REFERENCES barang(kodebarang) ON UPDATE CASCADE, FOREIGN
  KEY(kodepemasok) REFERENCES pemasok(kodepemasok) ON UPDATE
  CASCADE);
- CREATE TABLE keluar (nomorkeluar INT NOT NULL,
  tanggalkeluar DATE NOT NULL, kodebarang varchar(6),
  jumlahbarang jumlahbarangdomain NOT NULL, hargasatuan INT
  NOT NULL, PRIMARY KEY(nomorkeluar,tanggalkeluar), FOREIGN
  KEY(kodebarang) REFERENCES barang(kodebarang) ON UPDATE
  CASCADE);
```

4. Isi tabel-tabel tersebut dengan data sebarang minimal 3 baris untuk tiap-tiap tabel !
   Jawab:

```
- INSERT INTO kategori VALUES('SBC001','SABUN CUCI');
- INSERT INTO kategori VALUES('SBM001','SABUN MANDI');
- INSERT INTO kategori VALUES('DTJ001','DITERJEN');
- INSERT INTO barang VALUES('EKO002','SABUN
  EKONOMI','SBC001');
- INSERT INTO barang VALUES('LFB001','SABUN
  LIFEBOY','SBM001');
- INSERT INTO barang VALUES('ATC001','DITERJEN
  ATTACK','DTJ001');
- INSERT INTO pemasok VALUES('UNI001','UNILEVER','JAKARTA');
- INSERT INTO pemasok VALUES('WNG001','WINGS','JAKARTA');
- INSERT INTO pemasok VALUES('ABD001','ABADI','SURABAYA');
- INSERT INTO masuk VALUES('00001','07-13-
  2007','LFB001','UNI001','100','500000');
- INSERT INTO masuk VALUES('00002','10-23-
  2007','ATC001','UNI001','200','450000');
- INSERT INTO masuk VALUES('00003','4-20-
  2007','EKO002','ABD001','30','60000');
```

```
-  INSERT INTO keluar VALUES('1','08-20-
   2007','LFB001','50','50000');
-  INSERT INTO keluar VALUES('2','08-29-
   2007','LFB001','10','50000');
-  INSERT INTO keluar VALUES('3','11-09-
   2007','ATC001','80','45000');
```

5. Buat *query-query* SELECT:
   Jawab:
   - *menampilkan kode-kode barang, tanggalmasuk, dan jumlahbarang yang didatangkan dari salah satu pemasok tertentu (boleh memilih salah satu pemasok yang ada) !*

     ```
     SELECT kodebarang,tanggalmasuk,jumlahbarang FROM masuk
     WHERE kodepemasok='UNI001';
     ```

   - *menampilkan kode-kode barang dan hargasatuannya yang keluar dari tanggal 5 Juli 2007 hingga 10 Desember 2007 !*

     ```
     SELECT kodebarang,hargasatuan FROM keluar WHERE
     tanggalkeluar BETWEEN '07-05-2007' AND '12-10-2007';
     ```

   - *menampilkan kode-kode barang yang sudah pernah masuk tetapi belum pernah keluar !*

     ```
     SELECT kodebarang FROM masuk WHERE kodebarang NOT IN
     (SELECT kodebarang FROM keluar);
     ```

   - *menampilkan produk (cross product) tabel barang dan tabel kategori yang namabarang dan namakategorinya dimulai dengan huruf 'a' !*

     ```
     SELECT * FROM barang b CROSS JOIN kategori k WHERE
     b.namabarang LIKE 'a%' AND k.namakategori LIKE 'a%';
     ```

     Atau:

     ```
     SELECT * FROM barang b,kategori k WHERE b.namabarang LIKE
     'a%' AND k.namakategori LIKE 'a%';
     ```

   - *menampilkan seluruh kodebarang beserta kodepemasok dan namapemasoknya !*

     ```
     SELECT masuk.kodebarang, masuk.kodepemasok,
     pemasok.namapemasok FROM masuk, pemasok WHERE
     masuk.kodepemasok=pemasok.kodepemasok;
     ```

     Atau:

     ```
     SELECT masuk.kodebarang, masuk.kodepemasok,
     pemasok.namapemasok FROM masuk INNER JOIN pemasok ON
     masuk.kodepemasok=pemasok.kodepemasok;
     ```

- *menampilkan kode-kode barang dan jumlahbarang yang pemasoknya memiliki alamat yang mengandung kata 'Yogyakarta' !*

```
SELECT m.kodebarang, m.jumlahbarang FROM masuk m, pemasok
p WHERE m.kodepemasok=p.kodepemasok AND p.alamatpemasok
LIKE '%Yogyakarta%';
```

Atau:

```
SELECT m.kodebarang, m.jumlahbarang FROM masuk m, pemasok
p WHERE m.kodepemasok=p.kodepemasok AND p.alamatpemasok
CONTAINING 'Yogyakarta';
```

- *menampilkan nama-nama barang yang namapemasoknya mengandung huruf 'e'!*

```
SELECT b.namabarang FROM masuk m, barang b, pemasok p
WHERE m.kodebarang=b.kodebarang AND
m.kodepemasok=p.kodepemasok AND p.namapemasok LIKE '%e%';
```

Atau:

```
SELECT b.namabarang FROM masuk m, barang b, pemasok p
WHERE m.kodebarang=b.kodebarang AND
m.kodepemasok=p.kodepemasok AND p.namapemasok CONTAINING
'e';
```

- *menampilkan seluruh kodebarang beserta jumlahbarang masuk kumulatif untuk tiap-tiap barang tersebut !*

```
SELECT kodebarang, SUM(jumlahbarang) AS jumlahbarang FROM
barang GROUP BY kodebarang;
```

- *menampilkan tanggalkeluar dan namabarang yang jumlahbarang keluarnya paling besar !*

```
SELECT k.tanggalkeluar, b.namabarang FROM keluar k,barang
b WHERE k.kodebarang=b.kodebarang AND k.jumlahbarang IN
(SELECT MAX(jumlahbarang) FROM keluar);
```

Atau:

```
SELECT tanggalkeluar, namabarang FROM keluar JOIN barang
ON keluar.kodebarang=barang.kodebarang AND jumlahbarang IN
(SELECT MAX(jumlahbarang) FROM keluar);
```

6. Lakukan *backup database* di atas menjadi *file backup* dengan nama file barang.fbk!
   Jawab:

```
C:\Program Files\Firebird\Firebird_2_0\bin>gbak -v -t -user
SYSDBA -password masterkey d:\FDB\barang.fdb
d:\FDB\barang.fbk
```

KASUS 2: PENELITIAN DI PERGURUAN TINGGI

1. Tambahkan User:

```
gsec -user SYSDBA -password masterkey
add dexkai -pw dexkai -fname Decky -lname Hendarsyah
```

2. Modifikasi uid:

```
modify dexkai -uid 1
```

3. Buat database:

```
isql -user SYSDBA -password masterkey
CREATE DATABASE 'localhost:d:\dex\FDB\penelitian.fdb';
```

4. Buat tabel:
   - Tabel Teacher:

```
CREATE TABLE teacher(teacherid Varchar(5) NOT NULL, name
Varchar(20) NOT NULL, sex Char(1) NOT NULL, birthdate Date
NOT NULL, address Varchar(20), PRIMARY KEY(teacherid),
Check((sex='M') or (sex='m') or (sex='F') or (sex='f')));
```

   - Tabel Student:

```
CREATE TABLE student(studentid Varchar(5) NOT NULL, name
Varchar(20) NOT NULL, supervisor Varchar(5) NOT NULL,
enrollmentdate Date NOT NULL, graduationdate Date, PRIMARY
KEY(studentid), FOREIGN KEY(supervisor) REFERENCES
teacher(teacherid) ON UPDATE CASCADE,
Check(graduationdate>enrollmentdate));
```

   - Tabel Research:

```
CREATE TABLE research(researchid Varchar(5) NOT NULL,
title Varchar(20) NOT NULL, totalfund int, publishingdate
Date, PRIMARY KEY(researchid), Check(totalfund>0));
```

   - Tabel researchmember1:

```
CREATE TABLE researchmember1(researchid Varchar(5) NOT
NULL, memberid Varchar(5) NOT NULL, status Char(1),
PRIMARY KEY(researchid, memberid), FOREIGN KEY(researchid)
REFERENCES research(researchid) ON UPDATE CASCADE, FOREIGN
KEY(memberid) REFERENCES teacher(teacherid) ON UPDATE
CASCADE, Check((status='1') or (status='2')));
```

   - Tabel researchmember2:

```
CREATE TABLE researchmember2(researchid Varchar(5) NOT
NULL, memberid Varchar(5) NOT NULL, PRIMARY
KEY(researchid, memberid), FOREIGN KEY(researchid)
REFERENCES research(researchid) ON UPDATE CASCADE, FOREIGN
KEY(memberid) REFERENCES student(studentid) ON UPDATE
CASCADE);
```

5. Tambahkan 4 record masing2 table:
   - Tabel research:

```
- INSERT INTO research VALUES('R0001','BLUE
ENERGY',20000000,'03-03-2008');
- INSERT INTO research VALUES('R0002','KEUNGGULAN
FIREBIRD',1500000,'03-09-2008');
- INSERT INTO research VALUES('R0003','IMPLEMENTASI
WIMAX',14000000,'03-19-2008');
- INSERT INTO research VALUES('R0004','WIFI &
GPRS',12000000,'04-07-2008');
- INSERT INTO research VALUES('R0005','CITRA
BARCODE',1200000,'04-17-2008');
- INSERT INTO research VALUES('R0006','KELEMAHAN
SQL',1000000,'04-21-2008');
- INSERT INTO research VALUES('R0007','SISTEM INF.
AKADEMIK',1700000,'05-02-2008');
- INSERT INTO research VALUES('R0008','SISTEM INF.
GEOGRAFI',3400000,'05-22-2008');
```

   - Tabel teacher:

```
- INSERT INTO teacher VALUES('T0001','BUDI
HARTONO','M','08-08-1974','Yogyakarta');
- INSERT INTO teacher VALUES('T0002','RUDIANSYAH','M','09-
02-1972','Surabaya');
- INSERT INTO teacher VALUES('T0003','TETI
ENDANG','F','08-12-1979','BANDUNG');
- INSERT INTO teacher VALUES('T0004','PUTRI
UNAYA','F','08-12-1979','Padang');
```

   - Tabel student:

```
- INSERT INTO student VALUES('S0001','ANDIKA','T0001','08-
29-2005','03-19-2008');
- INSERT INTO student
VALUES('S0002','ARTIKASARI','T0003','08-29-2005','08-23-
2008');
- INSERT INTO student VALUES('S0003','JOKO
SUROTO','T0004','08-27-2004','08-23-2008');
- INSERT INTO student VALUES('S0004','KETTY','T0003','08-
25-2003','03-22-2007');
```

   - Tabel researchmember1:

```
- INSERT INTO researchmember1
VALUES('R0001','T0001','1');
- INSERT INTO researchmember1
VALUES('R0002','T0002','1');
- INSERT INTO researchmember1
VALUES('R0003','T0003','2');
- INSERT INTO researchmember1
VALUES('R0004','T0001','2');
```

- Tabel researchmember2:

```
- INSERT INTO researchmember2 VALUES('R0005','S0001');
- INSERT INTO researchmember2 VALUES('R0006','S0002');
- INSERT INTO researchmember2 VALUES('R0007','S0003');
- INSERT INTO researchmember2 VALUES('R0008','S0004');
```

6. Query-query:

```
● SELECT researchid, title FROM research WHERE totalfund>0;
● SELECT studentid, name FROM student WHERE (graduationdate-
  enrollmentdate)>1500;
● SELECT name FROM teacher WHERE upper(sex)='F' AND name
  CONTAINING 'a';
● SELECT student.name, teacher.name FROM student JOIN
  teacher ON student.supervisor=teacher.teacherid;
● SELECT name FROM student WHERE supervisor IN (SELECT
  memberid FROM researchmember1 WHERE
  student.supervisor=researchmember1.memberid);
● SELECT title FROM research,teacher,researchmember1 WHERE
  researchmember1.memberid=teacher.teacherid AND
  researchmember1.researchid=research.researchid AND
  teacher.name LIKE 'a%';
```

Dari tulisan di atas itulah yang dapat penulis jelaskan dan sampaikan, mungkin masih banyak kekurangan disana sini. Namun bagi pembaca yang berminat dengan Database Firebird, penulis berharap agar dapat juga berpartisipasi membuat tulisan mengenai Database Firebird dan mengembangkan tulisan-tulisan yang telah ada, karena penulis lihat masih banyak orang yang belum kenal dengan Database Firebird ini, terbukti dengan minimnya tulisan-tulisan atau artikel-artikel mengenai Database Firebird. Mudah-mudahan dengan adanya tulisan Database Firebird bag 1 dan 2 ini bisa memaju para penulis lain yang tertarik pada Database terutama Database Firebird.

REFERENSI:

1. Budi Raharjo, *"Langkah dan Proses Tercepat Menjadi Programmer Kylix & Delphi"*, Informatika Bandung, 2007.
2. Scotts Valey, *"Data defenition Guide"*, Intebase Software Corporation, 1998.
3. Scotts Valey, *"Language Reference"*, Intebase Software Corporation, 1998.
4. Scotts Valey, *"Programmer's Guide"*, Intebase Software Corporation, 1998.

Biografi Penulis

Decky Hendarsyah, lahir di Bukittinggi Sumatera Barat pada tahun 1978. SD sampai SMU ditempuh di Padang Panjang Sumatera Barat. Merupakan Alumni SMU Negeri 1 Padang Panjang, tamat tahun 1997. Kemudian melanjutkan pendidikan Komputer 1 tahun setingkat Diploma 1 (D1) di IPK Bukittinggi, tamat pada tahun 1998. Kuliah S1 di Universitas Putra Indonesia (UPI) "YPTK" Padang mengambil jurusan Sistem Informasi, lulus tahun 2002. Bekerja sebagai dosen dan Kepala UPT Puskom STIE Syari'ah Bengkalis. Pertengahan tahun 2008 melanjutkan pendidikan S2 di Megister Ilmu Komputer FMIPA UGM Yogyakarta. Menyukai kryptographi, database, pemrograman seperti bahasa pemrograman Pascal, Borland Delphi dan PHP. Sekarang sedang mempelajari dan ingin memperdalam bahasa pemrograman java dan juga tertarik pada GIS/SIG dan komunikasi data.