Lecture Notes

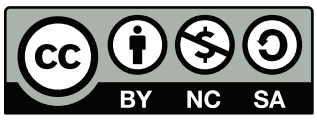
Database System
(Sistem Basis Data)

disusun oleh

Nanda Arista Rizki, M.Si.
Fidia Deny Tisna Amijaya, M.Si.

JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS MULAWARMAN
2019

HTTP://MATH.FMIPA.UNMUL.AC.ID



April, 2019

Daftar Isi

Daftar Isi **ii**

1 Pengenalan Sistem Basis Data **1**

1.1 Skala Pengukuran Data 2
1.2 Model *Entity-Relationship* 4
1.2.1 Entitas 4
1.2.2 Atribut 5
1.2.3 Relasi dan himpunan relasi 6
1.3 *Entity-Relationship Diagram* (ERD) 10
1.4 Aljabar Relasional 12
1.4.1 *Select* 12
1.4.2 *Project* 13
1.4.3 *Cartesian-product* 14
1.4.4 *Union* 15
1.4.5 *Set-difference* 16
1.4.6 *Rename* 17
1.5 Latihan 19

2 Pemrograman SQL **22**

2.1 Membuat *Database* 23
2.2 Mendefinisikan Tipe Data 28
2.3 Mengkalkulasi Atribut 28
2.4 Menggunakan Variabel dan Fungsi 30
2.5 Mengurutkan Data 32
2.6 Kriteria Seleksi 34
2.7 Logika *Boolean* 39
2.8 Logika Kondisi 42
2.9 Ringkasan Data 45

3 Implementasi Lazarus **48**

3.1 Komponen DBGrid 49

3.2 Komponen DBNavigator dan DBEdit 51

3.3 Memanipulasi *Database* 53

3.4 Memanfaatkan StringGrid 54

3.5 Menggunakan LazReport 55

Lampiran **65**

Materi Kuliah

Pertemuan	Bahan Kajian
1	Pengenalan
2	Model *Entity-Relationship* (entitas dan atribut)
3	Model *Entity-Relationship* (relasi dan himpunan relasi)
4	*Entity-Relationship Diagram* (ERD)
5	Aljabar relasional
6	*Structured Query Language* (SQL)
7	SQL
8	UTS
9	Manajemen dataset
10	Komponen dbgrid
11	Komponen dbedit dan dbnavigator
12	Manipulasi data
13	Komponen lazreport
14	Diskusi pembuatan program (PROYEK)
15	Diskusi pembuatan program (PROYEK)
16	UAS

Sistem Penilaian

Kriteria	Persentase	Keterangan
Afektif	10%	Kehadiran, Attitude
Praktikum	20%	Tugas, UTP, dan UAP (Asisten)
UTS	30%	Tugas, dan Nilai UTS
UAS	40%	Tugas, Diskusi proyek, dan Nilai UAS

Info lebih lanjut, kunjungi http://math.fmipa.unmul.ac.id/index.php/nanda.

BAB 1

Pengenalan Sistem Basis Data

Database-management System (DBMS) adalah kumpulan data yang saling terkait dan satu set program untuk mengakses datanya. Kumpulan data ini disebut basis data (*database*), yang merupakan kumpulan informasi mengenai fakta-fakta yang disimpan dalam komputer secara sistematik. Tujuan utama DBMS adalah menyimpan dan mencari informasi basis data dengan mudah, cepat, dan efisien. Sistem basis data dirancang untuk mengelola banyak informasi. Data-data ini perlu diolah melalui analisis tertentu sehingga berguna dalam pengambilan keputusan. Basis data sangat erat kaitannya dengan kehidupan sehari-hari, yaitu data perusahaan, data bank, universitas, dan lain-lain. Data tertentu juga dapat diperoleh melalui hasil pengamatan. Pemanfaatan basis data dilakukan dengan tujuan yaitu:

1. kecepatan dan kemudahan (*speed*)
2. efisiensi ruang penyimpanan (*space*)
3. keakuratan (*accuracy*)
4. ketersediaan (*availability*)
5. kelengkapan (*completeness*)
6. keamanan (*security*)
7. data dapat dipakai secara bersama (*shareability*).

Saat ini semua kebutuhan informasi dapat dipenuhi dengan cepat. Mulai mengetahui kurs dolar, jadwal tayang bioskop, banyaknya *subscriber* seorang *youtuber*, sampai banyaknya *follower* akun seseorang. Selain itu, di era industri 4.0 ini juga mempermudah seseorang dalam melakukan aktifitasnya, yaitu mulai dari memesan tiket, belanja bulanan, membayar uang kos dan lain-lain. Tanpa disadari, semua data informasi ini masuk ke dalam suatu sistem basis data. Sistem ini sangat diperlukan untuk analisis lebih lanjut mengenai semua hal yang akan diteliti. Dengan teknik *data mining*, *server* untuk sistem basis data dapat mengetahui pada tanggal berapa saja jadwal penerbangan padat, kapan waktu yang tepat untuk suatu toko memberi diskon belanja, dan kasus menarik lainnya.

Misalkan di kota Samarinda saat ini sangat viral dengan adanya penjualan roti. Jika pihak penjual membuat suatu aplikasi sehingga pembeli dapat mengakses menu pembelian, maka semua data penjualan yang tersimpan dapat dimanfaatkan oleh penjual untuk meningkatkan pendapatannya. Basis data yang tersimpan dalam *server* dapat berupa waktu pembelian, tanggal pembelian, cuaca saat pembelian, banyaknya pesanan, jenis roti yang dipesan, spesifikasi *handphone* pembeli, dan lain-lain. Bahkan jika pembeli memiliki *member card*, maka penjual roti dapat mengetahui identitas yang lebih banyak lagi untuk digali.

1.1 Skala Pengukuran Data

Sebelum mengenal lebih jauh tentang basis data dan penggunaannya, alangkah baiknya jika mempelajari skala pengukuran data terlebih dahulu. Data dibagi menjadi 4 skala, yaitu nominal, ordinal, interval, dan rasio yang dapat dirangkum dalam Tabel 1.1.

Nilai-nilai dari data nominal digunakan ketika data tersebut hanya sebagai pembeda saja. Dalam hal ini, semua data dianggap sama atau setara. Artinya tidak ada datum yang lebih unggul, lebih baik, dan lebih tinggi. Contoh data nominal adalah warna, jenis pekerjaan, jenis kelamin, program studi, dan lain-lain. Lebih mudah dipahami ketika nilai-nilai dari variabel nominal bernilai bilangan bulat. Misalkan warna kesukaan mahasiswa program studi matematika adalah merah marun, biru dongker, hijau toska, kuning langsat, dan putih susu. Peneliti boleh memberi nilai-nilai untuk setiap warna tersebut. Karena data ini bersifat nominal,

maka nilai-nilai warna ini tergantung oleh peneliti. Sebagai contoh, warna merah marun bernilai -2, biru dongker bernilai -1, hijau toska bernilai 0, kuning langsat bernilai 1, dan putih susu bernilai 2. Peneliti lain mungkin saja memiliki penilaian berbeda. Artinya nilai data nominal boleh ditukar.

Tabel 1.1: Skala-skala Pengukuran Data

Skala	Tipe data	Operasi	Fitur pembeda	Ukuran pemusatan data
Nominal	diskrit	$=, \neq$	hanya kategori	hanya modus
Ordinal	diskrit	$=, \neq, \leq, \geq$	kategori terurut	modus dan median
Interval	kontinu	$=, \neq, \leq, \geq, +, -$	makna interval	modus, median, dan mean
Rasio	kontinu	$=, \neq, \leq, \geq, +, -, \times, \div$	nilai nol mutlak	modus, median, dan mean

Selanjutnya adalah data ordinal. Dalam data ordinal, urutan data diperhatikan. Artinya, urutan nilai dari data ordinal harus monoton. Salah satu contoh data ordinal adalah tingkat kelezatan suatu makanan. Seperti halnya data nominal, Peneliti menginput jika nilai dari data ordinal diubah ke dalam bilangan bulat. Karena "sangat lezat" lebih baik daripada "lezat" dan "lezat" lebih baik daripada "hambar", maka Peneliti boleh menginput tingkat "hambar" bernilai 1, tingkat "lezat" bernilai 2, dan tingkat "sangat lezat" bernilai 3. Peneliti dilarang memberi nilai untuk tingkat "hambar" sebesar 3, jika nilai untuk tingkat "lezat" dan "sangat lezat" masing-masing adalah 1 dan 2.

Selanjutnya, bayangkan jika banyaknya nilai data ordinal sangat banyak lalu dikumpulkan menjadi bilangan yang padat seperti bilangan rasional. Semua nilai-nilai ini memiliki peningkatan yang konstan dan relatif kecil. Data ordinal ini dapat dikatakan data interval. Data interval berbeda dengan istilah selang interval pada matakuliah kalkulus. Data inverval lebih menekankan pada sifat dari himpunannya saja. Contoh yang paling mudah untuk mengilustrasikan data interval adalah data suhu (temperatur). Semakin tinggi nilai suhu suatu ruangan, maka keadaan ruangan tersebut semakin panas. Dalam hal ini, suhu $24^{\circ}C$ bukan berarti 2 kali lebih panas dari $12^{\circ}C$ dan suhu $12^{\circ}C$ bukan berarti 2 kali lebih panas dari $-12^{\circ}C$. Suhu ruangan yang tidak panas juga bukan berarti memiliki nilai suhu $0^{\circ}C$.

Skala pengukuran data yang paling tinggi adalah data rasio. Nilai untuk data rasio dapat dikonversi menjadi data interval, data ordinal, atau data nominal.

Perhatikan bahwa data interval juga dapat diubah menjadi data ordinal, atau data nominal. Hasil pengukuran data rasio yang bisa dibedakan, diurutkan, memiliki jarak tertentu, dan bisa dibandingkan. Nilai nol mutlak pada data rasio berarti benar-benar menyatakan tidak ada.

1.2 Model *Entity-Relationship*

Model *Entity-Relationship* memiliki dua komponen utama pembentuk, yaitu entitas (*entity*) dan relasi (*relation*). Kedua komponen ini dideskripsikan lebih jauh melalui sejumlah atribut atau properti.

1.2.1 Entitas

Entitas adalah individu yang mewakili sesuatu obyek yang nyata dan dapat dibedakan dari sesuatu yang lain. Contohnya adalah Winda, Ayla, Neni, Nurul, Mobil Toyota, Mobil Honda, Mobil Daihatsu, Melodi, Haruka, Buku aljabar linier, Buku basis data, Nasi kuning, Nasi goreng, Jus wortel, *Milkshake*, Roti balok, dan lain-lain.

Atribut Entitas

nim	nama_mhs	alamat_mhs	hobi_mhs	
1701	Anastasya	Jl. Anggur 1, Amuntai	vlogging, memasak, menulis	Entitas 1
1702	Bunga	Jl. Blueberry 2, Balikpapan		
1801	Citra	Jl. Cempedak 3, Cianjur	bermain musik	Entitas 3
1802	Dewi	Jl. Durian 4, Denpasar	travelling, hiking	
1803	Erlina	Jl. Elai 5, Edinburgh	fotografi, desain grafis	

Gambar 1.1: Himpunan entitas Mahasiswa

Himpunan entitas (*entity set*) adalah sekelompok entitas yang berada dalam lingkup yang sama. Contoh himpunan entitas antara lain Pegawai, Mobil, Pelanggan, Mahasiswa, Buku, Makanan, Minuman, dan lain sebagainya. Himpunan entitas (yang kurang praktis dalam penyebutannya) ini seringkali digantikan dengan sebutan entitas saja. Oleh karena itu, sering kali ditemukan penggunaan istilah enti-

tas yang berarti himpunan entitas. Dalam sistem basis data, istilah entitas merujuk pada nama tabel. Karena penggunaan entitas sebagai nama tabel, maka baris ke i dari entitas disebut data ke i. Perhatikan bahwa Gambar 1.1 merupakan entitas Mahasiswa.

1.2.2 Atribut

Komponen selanjutnya adalah atribut. Komponen atribut adalah karakteristik (properti) yang mendeskripsikan suatu entitas. Dalam sistem basis data, atribut ini merujuk pada kolom tabel. Sebagai contoh, atribut pada entitas Mahasiswa yang ditampilkan dalam Gambar 1.1 adalah nim, nama_mhs, alamat_mhs, dan hobi_mhs.

Atribut dibagi menjadi tujuh jenis yaitu atribut sederhana, atribut komposit, atribut bernilai tunggal, atribut bernilai banyak, atribut harus bernilai (*mandatory attribute*), atribut nilai *null*, dan atribut turunan. Beberapa atribut ditunjukkan dalam Gambar 1.2. Adapun penjelasan masing-masing jenis ini adalah sebagai berikut:

1. Atribut sederhana adalah atribut atomik yang tidak dapat dipilah lagi. Contohnya adalah jenis kelamin dan jurusan.

2. Atribut komposit adalah atribut yang masih dapat diuraikan lagi menjadi sub-sub atribut yang masing-masing memiliki makna. Sebagai contoh, pada atribut Nama dapat diuraikan menjadi nama depan, nama tengah, nama belakang. Contoh lainnya adalah atribut alamat (nama jalan, nomor rumah, kota).

3. Atribut bernilai tunggal adalah atribut yang hanya memiliki maksimal satu nilai di tiap datanya. Contohnya NIM, nama, dan tanggal lahir.

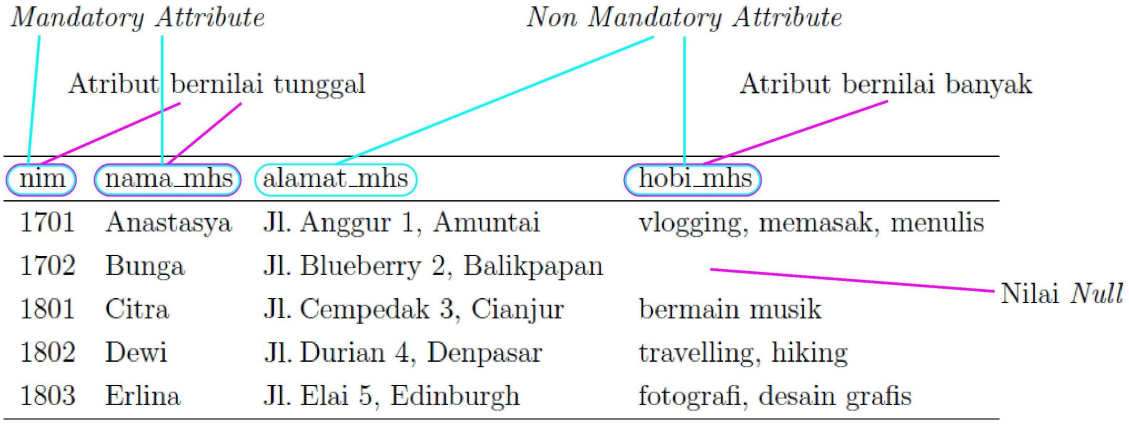
4. Atribut bernilai banyak adalah atribut yang dapat diisi lebih dari satu nilai. Misalnya *hobby* dan nomor *handphone*.

5. Atribut harus bernilai (*mandatory attribute*) adalah atribut yang harus berisi suatu data. Contohnya adalah NIM, nama mahasiswa, dan alamat.

6. Atribut nilai *null* adalah atribut yang belum memiliki nilai. Dalam hal ini, *null* artinya kosong. Contohnya, pada atribut hobi_mhs untuk entitas mahasiswa

(Gambar 1.2) bernama Bunga yang memang tidak memiliki *hobby*. Atribut *null* juga dapat diartikan bahwa datanya belum siap atau masih meragukan.

7. Atribut turunan adalah atribut yang nilainya dapat diturunkan dari atribut lainnya. Atribut turunan sebenarnya hanya digunakan sebagai penjelas dan dapat ditiadakan. Contoh atribut turunan adalah angkatan. Nilai-nilai pada atribut angkatan dapat diketahui dari atribut nim (biasanya dua karakter pertama dalam nim menyatakan dua digit bilangan tahun masuknya mahasiswa yang bersangkutan). Contoh atribut turunan lainnya adalah atribut indeks prestasi yang dapat diperoleh dari hasil perhitungan nilai.



Gambar 1.2: Beberapa jenis atribut

Primary key adalah atribut yang dapat membedakan data satu dengan data yang lainnya dalam suatu entitas. Contohnya adalah NIM, Kode_makanan, dan Kode_barang. Fungsi penggunaan *Primary Key* adalah untuk membedakan data satu dengan data yang lainnya.

1.2.3 Relasi dan himpunan relasi

Beberapa entitas yang berbeda dapat saling berkaitan dan dapat dihubungkan. Dalam sistem basis data, hubungan ini dinamakan relasi (*relationship*). Jadi, relasi menunjukkan adanya hubungan di antara sejumlah entitas yang berasal dari himpunan entitas yang berbeda. Contohnya adalah transaksi barang, Kartu Rencana Studi (KRS), Pembelian, dan lain sebagainya. Perhatikan Gambar 1.3, bahwa ma-

hasiswa dengan nim='1702' dan nama_mhs = 'Bunga' (dalam himpunan entitas Mahasiswa) memiliki relasi dengan entitas mata kuliah dengan kode_mk='MA02' dan nama_mk='Pengantar Topologi'. Relasi di antara kedua entitas tadi mengandung arti bahwa mahasiswa bernama Bunga sedang mempelajari mata kuliah Pengantar Topologi di sebuah program studi.

nim	nama_mhs	...	...
1701	Anastasya	...	...
1702	Bunga	...	...
1801	Citra	...	...
1802	Dewi	...	...
1803	Erlina	...	...

kode_mk	nama_mk	...	...
MA01	Sistem Basis Data	...	...
MA02	Pengantar Topologi	...	...
MA03	Proses Stokastik	...	...
MA04	Sistem Dinamik Kontinu	...	...
MA05	Pengantar Teori Modul	...	...

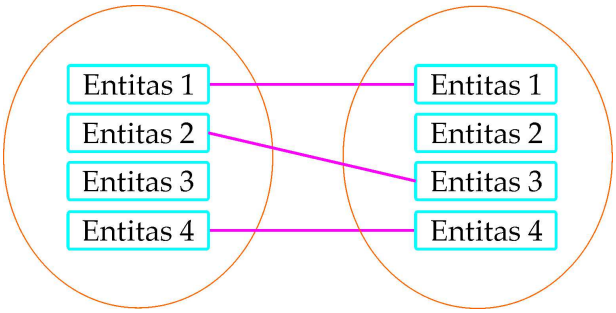
Gambar 1.3: Contoh relasi antara himpunan entitas Mahasiswa dan himpunan entitas mata kuliah

Himpunan relasi (*relationship sets*) merupakan kumpulan semua relasi dalam suatu sistem basis data. Seperti halnya penggunaan istilah himpunan entitas yang disingkat menjadi entitas (walaupun sebenarnya memiliki perbedaan makna), istilah himpunan relasi jarang sekali digunakan dan lebih sering disingkat dengan istilah relasi saja.

Perhatikan bahwa relasi antar entitas pada Gambar 1.3 hanya berlaku pada semester tertentu saja. Kelak akan ditunjukkan adanya faktor waktu dalam relasi antar entitas tersebut yang dapat diterapkan. Semua relasi yang ada di antara himpunan entitas tersebut membentuk sebuah himpunan relasi. Untuk menjelaskan apa yang terjadi di antara kedua himpunan relasi tersebut, maka dapat diberi nama himpunan relasi 'mempelajari' atau himpunan relasi 'belajar'.

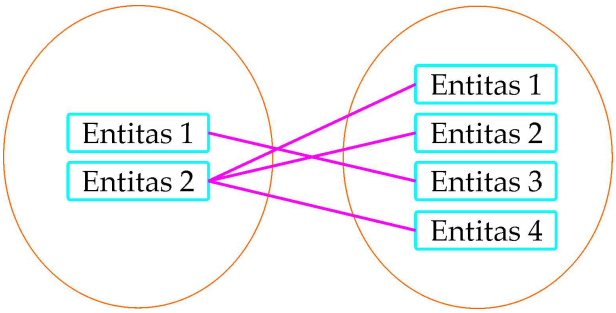
Kardinalitas atau derajat relasi adalah jumlah maksimum entitas yang dapat berelasi pada entitas lainnya. Jika diberikan dua himpunan entitas (yaitu A dan B), maka kardinalitas relasinya dibedakan menjadi 4 jenis, yaitu:

1. Satu ke Satu (*One to One*), yang berarti satu entitas A berhubungan paling banyak satu dengan entitas B. Begitupun sebaliknya, satu entitas pada himpunan B berhubungan paling banyak satu dengan entitas A. Perhatikan Gambar 1.4.



Gambar 1.4: Kardinalitas relasi Satu ke Satu

2. Satu ke Banyak (*One to Many*), yang berarti satu entitas A dapat berhubungan lebih dari satu dengan entitas B. Namun tidak berlaku sebaliknya, dimana setiap entitas pada himpunan entitas B berhubungan dengan paling banyak dengan satu entitas pada himpunan entitas A. Perhatikan Gambar 1.5.

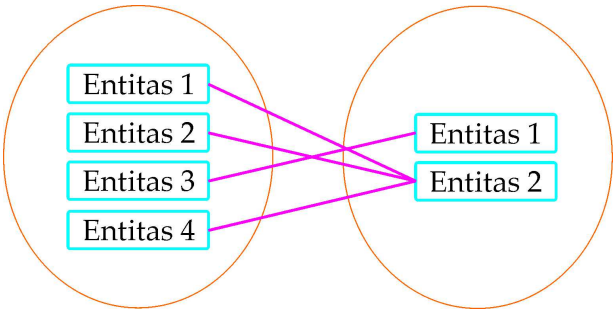


Gambar 1.5: Kardinalitas relasi Satu ke Banyak

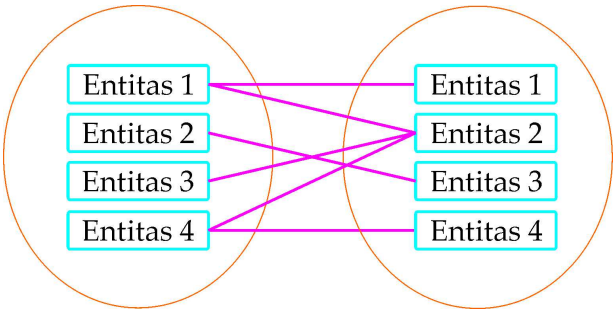
3. Banyak ke Satu (*Many to One*), yang berarti satu entitas A berhubungan paling banyak dengan satu entitas B. Namun tidak berlaku sebaliknya, dimana setiap entitas pada himpunan entitas B dapat berhubungan lebih dari satu dengan entitas A. Perhatikan Gambar 1.6.

   Kardinalitas relasi Satu ke Banyak dan Kardinalitas relasi Banyak ke Satu dapat dianggap sama, karena tinjauan kardinalitas relasi selalu dapat dilihat dari dua sisi. Artinya, posisi himpunan entitas A dan himpunan entitas B dapat saja ditukar.

4. Banyak ke Banyak (*Many to Many*), yang berarti satu entitas A dapat berhubungan lebih dari satu dengan entitas B dan sebaliknya. Perhatikan Gambar 1.7.



Gambar 1.6: Kardinalitas relasi Banyak ke Satu



Gambar 1.7: Kardinalitas relasi Banyak ke Banyak

Contoh kasus untuk relasi Satu ke Satu adalah satu dosen mengepalai satu jurusan di suatu universitas. Dalam hal ini seorang dosen paling banyak mengepalai satu jurusan, walaupun ada dosen yang tidak mengepalai jurusan manapun. Sebaliknya, setiap jurusan pasti dikepalai oleh paling banyak satu dosen. Selanjutnya diagram seperti pada Gambar 1.4 untuk kasus ini dapat disederhanakan menjadi Gambar 1.8.

Gambar 1.8: Penyederhanaan relasi Satu ke Satu

Contoh kasus untuk relasi Satu ke Banyak adalah satu dosen dapat mengajar banyak mata kuliah. Selanjutnya diagram seperti pada Gambar 1.5 untuk kasus ini dapat disederhanakan menjadi Gambar 1.9.

Gambar 1.9: Penyederhanaan relasi Satu ke Banyak

Contoh kasus untuk relasi Banyak ke Banyak adalah satu mahasiswa mempelajari

banyak mata kuliah, dan sebaliknya satu mata kuliah juga dipelajari oleh banyak mahasiswa. Selanjutnya diagram seperti pada Gambar 1.7 untuk kasus ini dapat disederhanakan menjadi Gambar 1.10.

Gambar 1.10: Penyederhanaan relasi Banyak ke Banyak

1.3 *Entity-Relationship Diagram* (ERD)

Model *Entity-Relationship* yang berisi komponen-komponen himpunan entitas dan himpunan relasi yang masing-masing dilengkapi dengan atribut-atribut, dapat digambarkan dengan lebih sistematis menggunakan Diagram *Entity-Relationship* atau *Entity-Relationship Diagram* (ERD). Notasi simbolik di dalam ERD yang wajib diketahui adalah sebagai berikut:

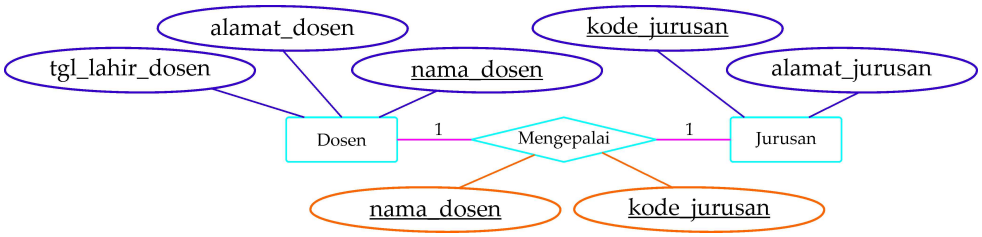
1. Persegi panjang, menyatakan himpunan entitas.
2. Elips, menyatakan atribut. Khusus untuk atribut yang berfungsi sebagai *primary key*, harus digarisbawahi.
3. Belah ketupat, menyatakan himpunan relasi.
4. Garis, sebagai penghubung antara himpunan relasi dengan himpunan entitas dan himpunan entitas dengan atributnya.
5. Kardinalitas relasi.

Berikut adalah contoh penggambaran relasi antar himpunan entitas beserta kardinalitas relasi dan atribut-atributnya:

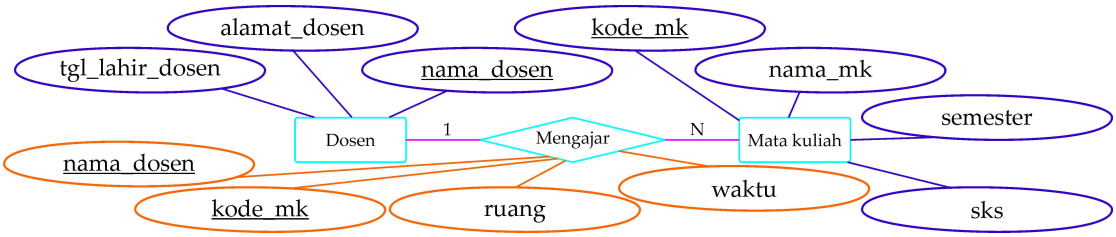
1. Relasi Satu ke Satu (*one-to-one*)
   Contoh ERD untuk relasi pada Gambar 1.8 adalah seperti pada Gambar 1.11a.
2. Relasi Satu ke Banyak (*one-to-many*)
   Contoh ERD untuk relasi pada Gambar 1.9 adalah seperti pada Gambar 1.11b.

3. Relasi Banyak ke Banyak (*many-to-many*)
   Contoh ERD untuk relasi pada Gambar 1.10 adalah seperti pada Gambar 1.11c.

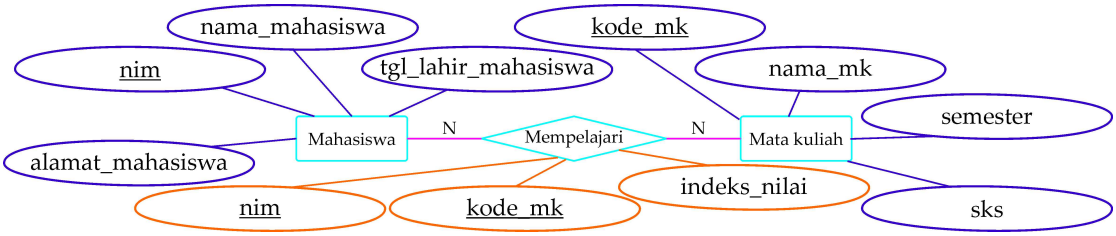
Penggabungan semua relasi untuk Gambar 1.11a, Gambar 1.11b, dan Gambar 1.11c ditunjukkan pada Lampiran 1.



(a) Relasi Satu ke Satu



(b) Relasi Satu ke Banyak



(c) Relasi Banyak ke Banyak

Gambar 1.11: Contoh ERD

Berikut adalah tahapan dalam perancangan sistem basis data:

1. Mempelajari studi kasus.
2. Membentuk himpunan entitas, himpunan relasi, dan atribut-atributnya.
3. Membuat ERD.
4. Membuat program.

5. Mengevaluasi sistem yang telah dibuat.

1.4 Aljabar Relasional

Bahasa Query merupakan bahasa yang termasuk dalam kategori bahasa tingkat tinggi (*high level language*) yang digunakan *user* untuk memperoleh informasi dari basis data. Bahasa Query dibagi menjadi dua jenis, yaitu bahasa prosedural dan bahasa non-prosedural. Pada sebuah bahasa prosedural, *user* meminta sistem untuk melakukan serangkaian operasi terhadap basis data dalam rangka mendapatkan informasi yang diinginkan. Namun dalam bahasa non-prosedural, *user* menunjukkan informasi yang diinginkan tanpa menyatakan suatu cara atau prosedur tertentu untuk memperoleh informasi tersebut.

Bahasa Query yang didasarkan pada operasi-operasi dalam aljabar relasional merupakan bahasa Query yang prosedural. Bahasa ini memiliki sejumlah operasi yang memanfaatkan suatu atau beberapa tabel atau relasi basis data sebagai masukan dan menghasilkan sebuah tabel atau relasi basis data yang baru sebagai keluarannya. Sejumlah operasi dasar yang dikenal dalam aljabar relasional, yaitu *select*, *project*, *cartesian-product*, *union*, *set-difference*, dan *rename*.

1.4.1 *Select* (σ)

Operasi ini digunakan untuk mengambil sejumlah baris data yang memenuhi predikat yang diberikan. Predikat mengacu pada kondisi yang ingin dipenuhi dalam operasi seleksi. Sintaks yang digunakan untuk menyatakan operasi ini adalah

$$\sigma_P(R) = \{t|t \in R, P(t)\}$$

dimana P adalah predikat pada atribut-atribut di R yang dapat dihubungkan dengan $\wedge$ (**and**), $\vee$ (**or**), dan $\neg$(**not**). Bentuk umum dari predikat seleksi adalah

$$< \text{atribut} > op < \text{atribut} >,$$

atau

$$< \text{atribut} > op < \text{konstanta} >,$$

dengan *op* adalah salah satu dari $=, \neq, >, \geq, <$, dan $\leq$. Sebagai contoh, perhatikan relasi r berikut

mk.nama	ketua_kelompok	nama_anggota	nilai_presentasi
Statistika Matematika	Rakhma	Mustika	81
Statistika Matematika	Rakhma	Rahayu	82
Model Risiko	Mustika	Mustika	83
Model Risiko	Rahayu	Rahayu	84

Maka operator seleksi $\sigma_{ketua\_kelompok=nama\_anggota \ \wedge \ nilai\_presentasi>82}(r)$ menghasilkan

mk.nama	ketua_kelompok	nama_anggota	nilai presentasi
Model Risiko	Mustika	Mustika	83
Model Risiko	Rahayu	Rahayu	84

Dalam relasi terakhir memuat atribut *mk.nama* yang berarti atribut *nama* yang berasal dari himpunan entitas *mk*.

1.4.2 *Project* (Π)

Operasi ini dapat menentukan atribut-atribut data dari sebuah tabel atau hasil Query yang akan ditampilkan. Dengan kata lain, operasi ini merupakan relasi antar beberapa kolom yang diperoleh dengan menghapus kolom yang tidak terdaftar. Sintaks yang digunakan untuk menyatakan operasi ini adalah

$$\prod_{A_1, A_2, \ldots, A_k}(r),$$

dengan $A_1, A_2, \ldots, A_k$ adalah nama-nama atribut dan r adalah relasinya. Dalam hal ini, diperbolehkan hanya satu atribut saja. Sebagai contoh, perhatikan relasi r berikut

mk.nama	mhs.nama	tugas
Rakhma	Statistika Matematika	84
Rakhma	Statistika Matematika	83
Rakhma	Model Risiko	82
Mustika	Statistika Matematika	81
Mustika	Statistika Matematika	80
Mustika	Model Risiko	79

Maka ekspresi *project* $\prod_{mk.nama,mhs.nama}(r)$ menghasilkan

mk.nama	mhs.nama
Rakhma	Statistika Matematika
Rakhma	Statistika Matematika
Rakhma	Model Risiko
Mustika	Statistika Matematika
Mustika	Statistika Matematika
Mustika	Model Risiko

Karena baris pertama adalah sama dengan baris kedua, sehingga hasilnya menjadi

mk.nama	mhs.nama
Rakhma	Statistika Matematika
Rakhma	Model Risiko
Mustika	Statistika Matematika
Mustika	Model Risiko

1.4.3 *Cartesian-product* ($\times$)

Operasi ini dapat menggabungkan data dari dua buah tabel atau hasil Query. Simbol yang digunakan untuk menyatakan operasi ini adalah "$\times$" dan sintaks yang digunakan untuk operasi ini adalah

$$r \times s = \{tq | t \in r \wedge q \in s\}.$$

Diasumsikan atribut $r(R)$ dan $s(S)$ adalah bebas atau dapat ditulis menjadi

$$R \cap S = \emptyset.$$

Jika atribut $r(R)$ dan $s(S)$ tidak saling bebas, maka perlu dilakukan *renaming*. Sebagai contoh, perhatikan relasi r berikut

mhs.nama	mhs.nim
Rakhma	65003
Rahayu	65015

dan relasi s berikut

mk.kode	ketua_kelompok	presentasi
001	Rakhma	78
001	Rahayu	79
002	Rakhma	80
003	Rakhma	81

Maka ekspresi $r \times s$ menghasilkan

mhs.nama	mhs.nim	mk.kode	ketua_kelompok	presentasi
Rakhma	65003	001	Rakhma	78
Rakhma	65003	001	Rahayu	79
Rakhma	65003	002	Rakhma	80
Rakhma	65003	003	Rakhma	81
Rahayu	65015	001	Rakhma	78
Rahayu	65015	001	Rahayu	79
Rahayu	65015	002	Rakhma	80
Rahayu	65015	003	Rakhma	81

Oleh karena itu, jika ekspresinya adalah $\sigma_{mhs.nama=ketua\_kelompok}(r \times s)$, maka menghasilkan

mhs.nama	mhs.nim	mk.kode	ketua_kelompok	presentasi
Rakhma	65003	001	Rakhma	78
Rakhma	65003	002	Rakhma	80
Rakhma	65003	003	Rakhma	81
Rahayu	65015	001	Rahayu	79

1.4.4 *Union* ($\cup$)

Operasi ini dapat menggabungkan data dari dua kelompok baris data (*row*) yang sejenis (memiliki hasil *project* yang sama). Notasi untuk operasi ini adalah

$$r \cup s = \{t | t \in r \vee t \in s\}.$$

Untuk semua $r \cup s$ harus memenuhi:

1. r dan s harus memiliki atribut yang sama
2. daerah asal atribut harus cocok tipenya. Misalnya, kolom kedua dari r harus mempunyai tipe yang sama dengan kolom kedua dari s.

Sebagai contoh, relasi r menyatakan keadaan sistem akademik dalam tahap pengajuan KRS berikut

mk.kode	mhs.nim
001	65001
001	65002
002	65001

dan relasi s yang menyatakan keadaan sistem dalam tahap perbaikan KRS berikut

mk.kode	mhs.nim
001	65002
002	65002

Maka operator *union* $r \cup s$ menghasilkan

mk.kode	mhs.nim
001	65001
001	65002
002	65001
002	65002

1.4.5 *Set-difference* $(-)$

Operasi ini merupakan kebalikan dari operasi *union*, yaitu pengurangan data di tabel atau hasil *project* pertama oleh data di tabel atau hasil *project* kedua. Simbol dari operasi ini adalah

$$r - s = \{t | t \in r \wedge t \notin s\}.$$

Untuk semua $r - s$ harus memenuhi:

1. r dan s harus memiliki atribut yang sama
2. daerah asal atribut harus cocok tipenya. Misalnya, kolom kedua dari r harus mempunyai tipe yang sama dengan kolom kedua dari s.

Sebagai contoh, relasi r menyatakan keadaan sistem akademik dalam tahap pengajuan KRS berikut

mk.kode	mhs.nim
001	65001
001	65002
002	65001

dan relasi s yang menyatakan keadaan sistem dalam tahap perbaikan KRS berikut

mk.kode	mhs.nim
001	65002
002	65002

Maka operator *union* $r - s$ menghasilkan

mk.kode	mhs.nim
001	65001
002	65001

1.4.6 *Rename* ($\rho_X(E)$)

Adakalanya operasi biner (*cartesian-product*, *set-difference*) diterapkan pada sumber data yang sama. Perlu diingat bahwaw suatu operator tidak boleh mengoperasikan tabel atau relasi yang sama. Operasi *rename* diperlukan ketika ingin melakukan penamaan kembali pada suatu tabel atau hasil relasi *project* agar dapat ditunjukkan acuan yang jelas dalam sebuah operasi yang lengkap, khususnya yang melibatkan dua atau lebih sumber data yang sama. Sintaks untuk operasi ini adalah

$$\rho_X(E),$$

dengan X menyatakan nama baru dari ekspresi E. Dalam hal ini, E adalah nama tabel. Misal diberikan himpunan entitas mahasiswa berikut

nomor_induk_mahasiswa	nama_mahasiswa
65001	Erlina
65003	Nurul
65004	Ghony

maka sintaks $\rho_{mhs}(mahasiswa)$ mengartikan bahwa himpunan entitas mahasiswa diberi nama baru yaitu himpunan entitas *mhs*. Dalam hal ini, fungsi operator *rename* adalah memberi nama tabel atau relasi yang panjang menjadi lebih pendek.

Operasi *rename* dapat pula diterapkan hingga ke level atribut, dengan sintaks

$$\rho_{X(A_1,A_2,...,A_n)}(r).$$

Artinya, atribut pertama dalam relasi r diberi nama A_1 dan seterusnya hingga atribut ke n dalam relasi r diberi nama A_n. Jadi sintaks ini tidak hanya memberi nama baru untuk tabel atau relasi, namun juga nama baru untuk setiap atributnya. Sebagai contoh, sintaks $\rho_{mhs(nim,nama)}(mahasiswa)$ menghasilkan relasi mhs berikut

nim	nama
65001	Erlina
65003	Nurul
65004	Ghony

Namun jika ingin mengganti beberapa atribut saja dalam relasi r, maka sintaks yang digunakan adalah

$$\rho_{a \longleftarrow b}(r),$$

sehingga atribut b dalam relasi r berubah nama menjadi atribut a.

Operator *rename* juga dapat menggabungkan relasi dari dua tabel yang terpisah. Perhatikan relasi smt_1 berikut

mk_smt_1	nama_mhs
Kalkulus I	Erlina
Kalkulus I	Nurul
Kalkulus I	Ghony
Pemrograman Komputer I	Erlina
Pemrograman Komputer I	Nurul
Pemrograman Komputer I	Ghony

dan relasi dalam smt_2 berikut

mk_smt_2	nama_mhs
Kalkulus II	Erlina
Kalkulus II	Nurul
Kalkulus II	Ghony
Pemrograman Komputer II	Erlina
Pemrograman Komputer II	Nurul
Pemrograman Komputer II	Ghony

Maka ekspresi $\rho_{mk \longleftarrow mk\_smt\_1}(smt\_1) \cup \rho_{mk \longleftarrow mk\_smt\_2}(smt\_2)$ menghasilkan

dan relasi berikut

mk	nama_mhs
Kalkulus I	Erlina
Kalkulus I	Nurul
Kalkulus I	Ghony
Pemrograman Komputer I	Erlina
Pemrograman Komputer I	Nurul
Pemrograman Komputer I	Ghony
Kalkulus II	Erlina
Kalkulus II	Nurul
Kalkulus II	Ghony
Pemrograman Komputer II	Erlina
Pemrograman Komputer II	Nurul
Pemrograman Komputer II	Ghony

Operator *rename* juga dapat menemukan nilai maksimum dalam suatu atribut. Sebagai contoh, perhatikan relasi $nilai\_pk\_2$ berikut

mk	nama_mhs	nilai
Pemrograman Komputer II	Erlina	80
Pemrograman Komputer II	Nurul	81
Pemrograman Komputer II	Ghony	82

Karena sintaks $\Pi_{nilai\_pk\_2.nilai}$ menghasilkan himpunan $\{80, 81, 82\}$ dan sintaks

$$\Pi_{nilai\_pk\_2.nilai}\left(\sigma_{nilai\_pk\_2.nilai<pk2.nilai}\left(nilai\_pk\_2 \times \rho_{pk2}(nilai\_pk\_2)\right)\right)$$

menghasilkan himpunan $\{80, 81\}$. Oleh karena itu, nilai maksimum dalam relasi $nilai\_pk\_2$ dapat ditentukan dengan sintaks

$$\Pi_{nilai\_pk\_2.nilai} - \Pi_{nilai\_pk\_2.nilai}\left(\sigma_{nilai\_pk\_2.nilai<pk2.nilai}\left(nilai\_pk\_2 \times \rho_{pk2}(nilai\_pk\_2)\right)\right).$$

1.5 Latihan

1. Berikut adalah data penjualan roti di suatu toko roti terkenal di Samarinda. Misal, diberikan 3 buah tabel yaitu tabel pelanggan, tabel kurir, dan tabel roti seperti yang disajikan pada Tabel 1.2, Tabel 1.3, dan Tabel 1.4.

Tabel 1.2: Pelanggan

id_pelanggan	nama_pelanggan	alamat_pelanggan	nomorhp_pelanggan
p1	Isyana	Jl. Perjuangan	081234567890
p2	Melodi	Jl. Pramuka	081234567891
p3	Raisa	Jl. M. Yamin	081234567892

Tabel 1.3: Kurir

id_kurir	nama_kurir	penilaian_pelanggan
k1	justin	4.4
k2	bieber	4.9

Tabel 1.4: Roti

id_roti	nama_roti	rasa_roti	harga_roti
r1	cheesecake	keju	18.181
r2	sweet choco	coklat	18.181
r3	matcha cake	green tea	14.545
r4	new vanila	vanila	14.545
r5	sugar-apple bread	srikaya	14.545

Peneliti boleh memberi kode untuk setiap informasi dalam tabel tersebut. Misal, pada tabel roti, untuk rasa keju diberi kode r1, rasa coklat diberi kode r2, rasa vanila diberi kode r3, rasa green tea diberi kode r4, dan rasa srikaya diberi kode r5. Selanjutnya apakah semua tabel tersebut dapat dihubungkan? Jika ya, apakah dengan suatu transaksi?

2. Tentukan entitas dari soal nomor 1. Lalu tentukan:

   (a) Atribut sederhana?

   (b) Atribut komposit?

   (c) Atribut bernilai tunggal?

   (d) Atribut bernilai banyak?

   (e) Atribut harus bernilai?

   (f) Atribut nilai *null*?

   (g) Atribut turunan?

3. Tentukan *primary key* tiap entitas dari soal nomor 1!

4. Berdasarkan soal nomor 1, tentukan kardinalitas dari setiap relasinya!
5. Berdasarkan soal nomor 1, buatlah diagram *Entity-Relationship* (ER) nya!
6. Berikan contoh penggunaan operasi *select*!
7. Berikan contoh penggunaan operasi *project*!
8. Berikan contoh penggunaan operasi *cartesian-product*!
9. Berikan contoh penggunaan operasi *union*!
10. Berikan contoh penggunaan operasi *set-difference*!
11. Berikan contoh penggunaan operasi *rename*!

BAB 2

Pemrograman SQL

Bahasa Query formal yang dipenuhi oleh ekspresi matematis yang telah dijelaskan pada Subbab 1.4, menjadi dasar teoritis dalam pembentukan bahasa Query terapan. Algoritma dan sintaks dari ekspresi-ekspresi bahasa Query terapan disusun berdasarkan bahasa Query formal.

Bahasa Query terapan yang paling populer adalah *Structured Query Language* (SQL). Bahasa ini sering diterapkan dalam berbagai *development tools* serta program aplikasi ketika berinteraksi dengan basis data. Bahasa SQL ini sering digunakan karena diakomodir oleh hampir semua *Relational Database Management System* (RDBMS), yaitu *software* pengelolaan basis data yang akan menentukan bagaimana data diorganisasi, disimpan, diubah, dan diambil kembali. Selain itu, *software* RDBMS juga dapat menerapkan mekanisme pengamanan data, pemakaian data secara bersama, pemaksaan akuratan/konsistensi data, dan sebagainya.

Software RDBMS yang digunakan dalam Bab 2 dan Bab 3 adalah Firebird. Firebird merupakan RDBMS yang bersifat *open source* yang dapat diunduh secara gratis di http://firebirdsql.org. *Firebird Interactive* SQL *Utility* atau disingkat ISQL merupakan alat interaktif berupa *command line* yang disediakan oleh Firebird untuk mengakses *database* Firebird. Namun untuk mempermudah dalam pengoperasian, dapat menggunakan *software* WFSQL karena sudah menyediakan fasilitas berbasis *Graphical User Interface* (GUI).

2.1 Membuat *Database*

Structured Query Language (SQL) merupakan bahasa yang tidak menganut *case sensitive*, sehingga perbedaan dalam penggunaan huruf besar atau kecil tidak berpengaruh. Langkah awal untuk memulai pemrograman SQL ini adalah membuat *database*. Adapun sintaksnya untuk membuat *database* adalah sebagai berikut:

```
1 CREATE database "C:\direktori\db.fdb"
2 user 'sysdba' password 'masterkey';
```

Untuk mengkoneksikan suatu *database* yang telah dibuat, gunakan sintaks berikut:

```
1 CONNECT "C:\direktori\db.fdb"
2 user 'sysdba' password 'masterkey';
```

Setelah itu, untuk menampilkan informasi suatu *database*, gunakan sintaks berikut:

```
1 SHOW DATABASE;
```

Untuk menampilkan nama dari SQL *roles*, gunakan sintaks berikut:

```
1 SHOW ROLES;
```

Untuk menampilkan informasi mengenai versi *software* ISQL atau WFSQL, gunakan sintaks berikut:

```
1 SHOW VERSION; -- atau SHOW VER;
```

Perhatikan bahwa *username* dan *password* yang sering digunakan masing-masing adalah 'SYSDBA' dan 'masterkey'. *Username* dan *password* tersebut merupakan penggunaan standar. Untuk urusan administrasi pengguna, gunakan *command line* yang disediakan oleh Firebird yaitu **GSEC** yang hanya dapat dijalankan oleh *user* SYSDBA. Fasilitas ini juga dapat dijalankan melalui *Command Prompt*.

Untuk menambahkan *user* Nyaman Beneh sebagai *user* jajak dengan *password* "amplang", gunakan sintaks berikut:

```
1 gsec -user SYSDBA -password masterkey
2 GSEC> add jajak -pw amplang -fname Nyaman -lname Beneh
3 GSEC> quit
```

Perlu diperhatikan bahwa penggunaan tanda minus berbeda dengan tanda minus ganda (yang berarti suatu komentar). Untuk mengubah *password* untuk *user* jajak menjadi "samarinda", gunakan sintaks berikut:

```
1 gsec -user SYSDBA -password masterkey
2 GSEC> modify jajak -pw samarinda
3 GSEC> quit
```

Oleh karena itu, mengubah *password* untuk SYSDBA dari "masterkey" menjadi "unmul", gunakan sintaks berikut:

```
1 gsec -user SYSDBA -password masterkey -modify sysdba -pw unmul
```

Untuk menghapus *user* jajak pada server lokal, gunakan sintaks berikut:

```
1 gsec -user SYSDBA -password masterkey -delete jajak
```

Tabel 2.1: NamaEntitas

Atribut_1	Atribut_2	$\cdots$	Atribut_n
datake_1_atribut_1	datake_1_atribut_2	$\cdots$	datake_1_atribut_n
$\vdots$	$\vdots$	$\ddots$	$\vdots$
datake_i_atribut_1	datake_i_atribut_2	$\cdots$	datake_i_atribut_n
$\vdots$	$\vdots$	$\ddots$	$\vdots$
datake_m_atribut_1	datake_m_atribut_2	$\cdots$	datake_m_atribut_n

Setelah membuat atau mengkoneksi *database*, selanjutnya adalah membuat tabel entitas. Struktur *database* secara umum dapat dilihat pada Gambar (2.1). Berikut adalah sintaks untuk membuat tabel entitas:

```
CREATE TABLE nama_entitas (
    atribut_1 tipedata_1,
    atribut_2 tipedata_2,
    ...,
    atribut_n tipedata_n);
```

Lalu *input* data dalam entitas dengan sintaks berikut:

```
INSERT INTO nama_entitas VALUES (
    datake_i_atribut_1,
    datake_i_atribut_2,
    ...,
    datake_i_atribut_1);
```

Kesalahan peng-*input*-an data tidak dapat dihindari. Salah satu penyebabnya adalah *human error*. Prosedur untuk memperbaharui data yang spesifik berdasarkan atribut adalah menggunakan sintaks berikut:

```
UPDATE entitas
SET atribut_1=ekspresi_1,
atribut_2=ekspresi_2,
...,
atribut_n=ekspresi_n
WHERE kondisi;
```

Klausa **DELETE** digunakan untuk penghapusan beberapa data. Berikut adalah sintaks umum penggunaannya:

```
DELETE
FROM entitas
WHERE kondisi;
```

Untuk menghapus semua data dalam entitas, gunakan sintaks berikut:

```
DELETE
FROM entitas;
```

Kemudian untuk menghapus entitas beserta datanya dari memori, gunakan sintaks berikut:

```
1 DROP table entitas;
```

Perubahan yang dilakukan pada *database* dengan memanipulasi data (yakni perintah **INSERT**, **UPDATE**, dan **DELETE**) tidaklah permanen sampai perintah **COMMIT** dijalankan. Oleh karena itu, gunakan sintaks berikut untuk menjalankan perubahan data yang dimanipulasi.

```
1 COMMIT;
```

Perintah **COMMIT** biasanya dilakukan setelah semua pekerjaan selesai, lalu proses tersebut akan membuat suatu *savepoints*. Sebaliknya jika ingin mengembalikan pekerjaan ke *savepoints* terakhir, maka gunakan perintah **ROLLBACK**.

Untuk menampilkan nama-nama entitas yang ada dalam suatu *database*, dapat menggunakan sintaks berikut:

```
1 SHOW TABLES;
```

Untuk menampilkan suatu atribut dan tipe datanya, dapat menggunakan sintaks berikut:

```
1 SHOW TABLES entitas;
```

Selanjutnya untuk menampilkan data berdasarkan suatu atribut, dapat menggunakan sintaks berikut:

```
1 SELECT Atribut
2 FROM entitas;
```

Perhatikan bahwa klausa **SELECT** berasosiasi dengan operator projeksi dalam bahasa Query formal, yaitu $\prod_{Atribut}(entitas)$. Jika ingin menampilkan semua data dalam entitas, maka gunakan sintaks berikut:

```
1 SELECT *
2 FROM entitas;
```

Contoh *database* dapat dilihat pada Tabel (2.2). *Database* tersebut menggunakan Mahasiswa sebagai nama entitasnya. **NULL** pada data ke 1 dan data ke 4 merupakan karakter kosong. Berikut adalah sintaks dalam membuat entitasnya:

```
1 CREATE TABLE Mahasiswa (
2 -- untuk komentar dapat menggunakan double dash
3 NIM VARCHAR(10) PRIMARY KEY NOT NULL, -- ini komentar kedua
4 NamaDepan VARCHAR(100), /* ini komentar ketiga */
5 NamaBelakang VARCHAR(100));
```

Tabel 2.2: Mahasiswa

NIM	NamaDepan	NamaBelakang
1707065001	Erlina	
1707065003	Nurul	Rakhmawaty
1707065004	Ghony	Nurhuda
1707065010	Ersin	
1707065011	Vika	Novitasari

Sangat memungkinkan jika suatu atribut dapat mengaitkan dua atau lebih tabel entitas. Misalnya, ketika ingin mengaitkan atribut yang sama dari dua entitas berbeda. Tabel-tabel yang menjadi sumber Query harus memiliki keterhubungan (relasi). Ekspresi ***Entitas1.Atribut_A*** menunjukkan bahwa *Atribut_A* berasal dari tabel entitas *Entitas1*. Sebagai contoh, entitas *mhs* yang terdiri dari atribut *nim*, *nama*, dan *alamat*; dan entitas *nilai* yang terdiri dari atribut *nim*, *tugas*, *uts* dan *uas*. Jika ingin menampilkan nama mahasiswa beserta nilai tugasnya, maka sintaks dalam SQL adalah sebagai berikut

```
1 SELECT
2 mhs.nama,
```

```
3 nilai.tugas
4 FROM mhs, nilai
5 WHERE mhs.nim=nilai.nim;
```

Ekspresi SQL ini merupakan bentuk terapan dari ekspresi formal dalam bahasa Query formal berikut

$$\prod_{nama,tugas} (\sigma_{mhs.nim=nilai.nim}(mhs \times nilai)).$$

Klausa **WHERE** dan kriteria seleksi lainnya akan dijelaskan lebih lanjut pada Subbab 2.6.

2.2 Mendefinisikan Tipe Data

Tipe data harus didefinisikan untuk setiap atribut dalam entitas. Tipe data yang mendukung adalah tipe bilangan, tipe tanggal dan waktu, tipe karakter, tipe **BLOB** dan *array*. Lebih lanjut mengenai tipe data, dapat dilihat pada Lampiran 2.

2.3 Mengkalkulasi Atribut

Perhatikan Tabel (2.3), lalu ikuti sintaks berikut.

```
1 SELECT
2 'Namanya:',
3 NamaDepan
4 FROM Mahasiswa;
```

Selanjutnya bandingkan dengan sintaks berikut.

```
1 SELECT
2 5,
3 NamaDepan
4 FROM Mahasiswa;
```

Tabel 2.3: Data Mahasiswa dengan Kolom Numerik

NIM	NamaDepan	NamaBelakang	Kegemaran	NilaiUTS	NilaiUAS
1707065001	Erlina		Olahraga	77	86
1707065003	Nurul	Rakhmawaty	Musik	78	85
1707065004	Ghony	Nurhuda	Olahraga	79	84
1707065010	Ersin		Tari	80	83
1707065011	Vika	Novitasari	Vlog	81	82

Pemprograman SQL juga dapat melakukan perhitungan aritmatik ($+$, $-$, $*$, dan $/$) pada satu atau lebih atribut dalam suatu entitas. Perhatikan sintaks berikut:

```
1 SELECT
2 NIM,
3 NilaiUTS,
4 NilaiUAS,
5 0.4*NilaiUTS + 0.6*NilaiUAS
6 FROM Mahasiswa;
```

Untuk merangkai (*concatenate*) beberapa *string*, gunakan operator $\|$ ("*double-pipe*"). Perhatikan sintaks berikut.

```
1 SELECT
2 NIM,
3 NamaDepan || ' suka ' || Kegemaran
4 FROM Mahasiswa;
```

Bandingkan sintaks berikut:

```
1 SELECT
2 NIM,
3 NamaDepan || ' suka ' || Kegemaran AS Keterangan
4 FROM Mahasiswa;
```

2.4 Menggunakan Variabel dan Fungsi

Ada beberapa variabel yang dapat digunakan untuk mengetahui tanggal/waktu. Perhatikan sintaks berikut:

```
1  select current_timestamp from rdb$database;
2  select current_time from rdb$database;
3  select current_date from rdb$database;
4  select date 'today' from rdb$database;
5  select timestamp 'TODAY' from rdb$database;
6  select date 'Now' from rdb$database;
7  select time 'now' from rdb$database;
8  select timestamp 'NOW' from rdb$database;
9  select date 'Tomorrow' from rdb$database;
10 select timestamp 'TOMORROW' from rdb$database;
11 select date 'Yesterday' from rdb$database;
12 select timestamp 'YESTERDAY' from rdb$database;
```

Sintaks tersebut menggunakan **rdb$database** yang dapat menyimpan informasi dasar tentang *database*. **rdb$database** adalah entitas sistem yang selalu ada dalam setiap *database* Firebird dan dijamin hasilnya berupa 1 baris. Entitas ini sering menjadi latihan standar oleh para programer Firebird.

Fungsi yang dipelajari adalah fungsi kalimat, fungsi matematik, dan fungsi konversi (*cast*). Fungsi kalimat terdiri dari **CHAR_LENGTH** atau **CHARACTER_LENGTH**, **LEFT**, **LOWER**, **RIGHT**, **TRIM**, dan **UPPER**. Perhatikan sintaks berikut:

```
1 Select
2 CHAR_LENGTH('Sistem Basis Data') AS COBA1,
3 CHARACTER_LENGTH('Sistem Basis Data') AS COBA2,
4 LEFT('Sistem Basis Data',1) AS Coba3,
5 LOWER('Sistem Basis Data') AS Coba4,
6 RIGHT('Sistem Basis Data',2) AS Coba5,
7 TRIM('    Sistem Basis Data     ') AS Coba6,
8 UPPER('Sistem Basis Data') AS Coba7
9 from rdb$database;
```

Fungsi numerik terdiri dari **ABS**, **ACOS**, **ASIN**, **ATAN**, **CEIL** atau **CEILING**, **COS**, **COSH**, **COT**, **EXP**, **FLOOR**, **LN**, **LOG10**, **MOD**, **PI**, **POWER**, **RAND**, **ROUND**, **SIGN**, **SIN**, **SINH**, **SQRT**, **TAN**, dan **TANH**. Perhatikan sintaks berikut:

```
1  SELECT
2  ABS(-1),
3  ACOS(0.5),
4  ASIN(0.5),
5  ATAN(1),
6  CEIL(2.5),
7  CEILING(2.5),
8  FLOOR(2.5),
9  MOD(8,3),
10 PI(),
11 POWER(25,0.5),
12 RAND(),
13 ROUND(PI(), 2),
14 SIGN(-8)
15 from rdb$database;
```

Tabel 2.4: Beberapa Kemungkinan untuk Perintah **CAST**

Sumber	Tujuan
Tipe numerik	Tipe numerik, [VAR]CHAR, BLOB
[VAR]CHAR, BLOB	[VAR]CHAR, BLOB, Tipe numerik, DATE, TIME, TIMESTAMP
DATE, TIME	[VAR]CHAR, BLOB, TIMESTAMP
TIMESTAMP	[VAR]CHAR, BLOB DATE, TIME

Fungsi konversi digunakan untuk mengubah suatu tipe data menjadi tipe data yang diinginkan. Fungsi konversi menggunakan perintah **CAST**. Beberapa kemungkinan yang dipakai untuk perintah **CAST** dapat dilihat pada Tabel (2.4). Perhatikan sintaks berikut:

```
select date 'today' from rdb$database;
select date '2017-06-30' from rdb$database;
select '30' || '-June-' || '2017' from rdb$database;
select cast('2017-06-30' as varchar(20))from rdb$database;
select cast ('30' || '-June-' || '2017' as date) from rdb$database;
select CAST(123.456789 as SMALLINT) from rdb$database;
select CAST('9' as SMALLINT) from rdb$database;
select CAST(9 as varchar(2)) from rdb$database;
```

Selanjutnya perhatikan Tabel (2.5). Untuk mengganti **NULL** pada Tabel (2.5), gunakan sintaks berikut:

```
SELECT
Nilai_Paper,
ISNULL(CAST(Presentasi AS VARCHAR),'Unknown') AS Presentasi
FROM Nilai_Presentasi;
```

Tabel 2.5: Nilai_Presentasi

Urutan_Maju	Nilai_Paper	Presentasi
1	77	
2	78	75
3	78	66
4	79	68

2.5 Mengurutkan Data

Ketika pernyataan **SELECT** dieksekusi, maka hasilnya sering tidak terurut. Yang terjadi adalah baris-baris yang muncul diurutkan secara kronologi. Secara sederhananya, urutannya akan sama dengan urutan penggunaan pernyataan **INSERT**. Untuk mengurutkan data berdasarkan alfabet, gunakan klausa **ORDER BY** ke statemen **SELECT**. Sebagai contoh, sintaks berikut menggunakan data pada Tabel (2.2).

```
    SELECT
    NamaDepan,
```

```
3   NamaBelakang
4   FROM Mahasiswa
5   ORDER BY NamaBelakang;
```

Silahkan coba sintaks berikut dan perhatikan hasilnya.

```
1 SELECT
2 NamaBelakang,
3 NamaDepan
4 FROM Mahasiswa
5 ORDER BY NamaDepan;
```

Sintak tersebut juga sama dengan sintaks berikut:

```
1 SELECT
2 NamaBelakang,
3 NamaDepan
4 FROM Mahasiswa
5 ORDER BY NamaDepan ASC;
```

Kata kunci **ASC** (singkatan dari *ascending*) bersifat opsional. Untuk urutan menurun, gunakan kata kunci **DESC** (singkatan dari *descending*). Sebagai contoh:

```
1 SELECT
2 NamaDepan,
3 NamaBelakang
4 FROM Mahasiswa
5 ORDER BY NamaDepan DESC;
```

Untuk mengurutkan secara *Multiple Columns*, perhatikan contoh berikut:

```
1 SELECT
2 NamaDepan,
3 NamaBelakang
4 FROM Mahasiswa
5 ORDER BY NamaDepan, NamaBelakang;
```

Untuk mengurutkan atribut yang dikalkulasi, perhatikan sintaks berikut:

```
1 SELECT
2 NamaBelakang || ', ' || NamaDepan AS Nama
3 FROM Mahasiswa
4 ORDER BY Nama;
```

2.6 Kriteria Seleksi

Penyeleksian dalam SQL dimulai dengan klausa **FIRST**, yang digunakan untuk membatasi banyaknya baris dari himpunan berurut. Perhatikan kembali Tabel (2.3). Misalkan hanya dua baris pertama dari data yang akan ditampilkan, gunakan sintaks berikut:

```
1 SELECT
2 FIRST 2
3 NamaDepan AS Nama_Mahasiswa,
4 NilaiUAS AS Nilai_yang_diperoleh
5 FROM Mahasiswa;
```

Selanjutnya, klausa **SKIP** digunakan untuk menyembunyikan beberapa baris pertama. Misalkan menampilkan data ke tiga sampai terakhir, gunakan sintaks berikut:

```
1 SELECT
2 SKIP 2
3 NamaDepan AS Nama_Mahasiswa,
4 NilaiUAS AS Nilai_yang_diperoleh
5 FROM Mahasiswa;
```

Jika hanya ingin menampilkan dua baris terakhir dari data, gunakan sintaks berikut:

```
1 SELECT
2 SKIP ((select count(*) - 2 from Mahasiswa))
3 NamaDepan AS Nama_Mahasiswa,
```

```
4 NilaiUAS AS Nilai_yang_diperoleh
5 FROM Mahasiswa;
```

Klausa **FIRST** dan **SKIP** dapat digunakan bersamaan. Misalkan ingin menampilkan baris ke tiga sampai ke lima dari Tabel (2.3), gunakan sintaks berikut:

```
1 SELECT
2 FIRST 3 SKIP 2
3 NamaDepan AS Nama_Mahasiswa,
4 NilaiUAS AS Nilai_yang_diperoleh
5 FROM Mahasiswa;
```

Klausa penyeleksian berikutnya adalah **WHERE**. Klausa ini menyeleksi subhimpunan baris. Misalkan ingin melihat Mahasiswa yang memiliki kegemaran olahraga pada Tabel (2.3), gunakan sintaks berikut:

```
1 SELECT
2 NamaDepan,
3 NamaBelakang,
4 Kegemaran
5 FROM Mahasiswa
6 WHERE Kegemaran = 'Olahraga';
```

Selanjutnya, operator-operator perbandingan untuk klausa **WHERE** yang dapat digunakan adalah operator-operator =, <>, ! =, ∼=, ^ =, <, <=, >, >=, ! <, ∼<, ^ <, ! >, ∼> dan ^ >. Operator <>, ! =, ∼=, dan ^ = menyatakan perbandingan tidak sama. Operator ! <, ∼<, dan ^ < menyatakan perbandingan tidak lebih kecil. Sedangkan Operator ! >, ∼> dan ^ > menyatakan perbandingan tidak lebih besar. Perhatikan sintaks berikut:

```
1 SELECT
2 NamaDepan,
3 NamaBelakang,
4 NilaiUTS
5 FROM Mahasiswa
6 WHERE NilaiUTS ^< 80;
```

Meskipun jarang digunakan, operator < juga dapat digunakan pada kolom teks.

```
1 SELECT
2 NamaDepan,
3 NamaBelakang,
4 Kegemaran
5 FROM Mahasiswa
6 WHERE Kegemaran < 'N';
```

Klausa **WHERE** berasosiasi dengan operasi seleksi dalam bahasa Query formal. Oleh karena itu, sintaks berikut

```
1 SELECT *
2 FROM mhs
3 WHERE nim='001';
```

ekivalen dengan operasi seleksi $\sigma_{nim='001'}(mhs)$.

Selain operator-operator perbandingan, klausa **WHERE** juga dapat menggunakan predikat-predikat perbandingan yang meliputi **LIKE**, **STARTING WITH**, **CONTAINING**, **BETWEEN**, **IS [NOT] NULL IS [NOT] DISTINCT FROM**, dan **SIMILAR TO**. Operator predikat **STARTING WITH** digunakan mencari data bertipe *string* atau *string-like* yang diawali dengan karakter tertentu. Operator predikat **CONTAINING** digunakan mencari data bertipe *string* atau *string-like* yang mengandung rangkaian karekter tertentu. Operator **CONTAINING** tidak bersifat *case-sensitive*. Operator predikat **BETWEEN** digunakan untuk mencari data yang memiliki nilai yang terletak di antara interval tertentu. Operator predikat **SIMILAR TO** membandingkan suatu *string* dengan suatu pola ekspresi reguler SQL. Operator predikat **SIMILAR TO** menghasilkan *output* yaitu **TRUE** atau **FALSE**. Perhatikan sintaks berikut:

```
1 Select
2 'Apple' similar to 'Apple', --true
3 'Apples' similar to 'Apple', --false
4 'Birne' similar to 'B_rne', --true
5 'Birne' similar to 'B_ne', --false
```

```
'Birne' similar to 'B%ne', --true
'Birne' similar to 'Bir%ne%', --true
'Citroen' similar to 'Cit[arju]oen', -- true
'Citroen' similar to 'Ci[tr]oen', -- false
'Citroen' similar to 'Ci[tr][tr]oen', -- true
'Datte' similar to 'Dat[q-u]e', -- true
'Datte' similar to 'Dat[abq-uy]e', -- true
'Datte' similar to 'Dat[bcg-km-pwz]e' -- false
from rdb$database;
```

Tabel 2.6: Movies

MovieID	MovieTitle	Rating
1	Love Actually	Rating
2	North by Northwest	Not Rated
3	Love and Death	PG
4	The Truman Show	PG
5	Everyone Says I Love You	Rating
6	Down with Love	PG-13
7	Finding Nemo	G

Operator predikat **LIKE** membandingkan ekspresi bertipe karakter dengan pola yang didefinisikan dalam ekspresi tertentu. Operator predikat ini bersifat *case-sensitive*. Misalkan untuk mencocokkan pola data pada Tabel (2.6), gunakan operator **LIKE**. Perhatikan sintaks berikut:

```
SELECT
MovieTitle AS Movie
FROM Movies
WHERE MovieTitle LIKE '%LOVE%';
```

Bandingkan dengan sintaks berikut:

```
SELECT
MovieTitle AS Movie
FROM Movies
WHERE MovieTitle LIKE 'LOVE%';
```

atau

```
SELECT
MovieTitle AS Movie
FROM Movies
WHERE MovieTitle LIKE '%LOVE';
```

atau

```
SELECT
MovieTitle AS Movie
FROM Movies
WHERE MovieTitle LIKE '% LOVE %';
```

Tabel 2.7: Actors

ActorID	FirstName	LastName
1	Cary	Grant
2	Mary	Steenburgen
3	Jon	Voight
4	Dustin	Hoffman
5	John	Wayne
6	Gary	Cooper

Tanda persen (%) merupakan *wildcard* yang sering digunakan. Namun dalam praktiknya, ada beberapa kemungkinan yang muncul. Untuk menyeleksi dapat juga menggunakan *wildcard* berupa garis bawah (_), karakter dalam kurung siku, dan karakter dalam kurung siku ditambah simbol *caret* (^). Perhatikan Tabel (2.7) lalu ikuti sintaks berikut:

```
SELECT
FirstName,
LastName
FROM Actors
WHERE FirstName LIKE '_ARY';
```

Bandingkan dengan sintaks berikut:

```
SELECT
```

```
2 FirstName,
3 LastName
4 FROM Actors
5 WHERE FirstName LIKE 'J_N';
```

Untuk penggunaan karakter dalam kurung siku, perhatikan sintaks berikut:

```
1 SELECT
2 FirstName,
3 LastName
4 FROM Actors
5 WHERE FirstName LIKE '[CM]ARY';
```

Bandingkan dengan sintaks berikut:

```
1 SELECT
2 FirstName,
3 LastName
4 FROM Actors
5 WHERE FirstName LIKE '[^CM]ARY';
```

Untuk mencari data yang menggunakan karakter *underscore*, dapat memanfaatkan karakter "#" yang dispesifikasi sebagai karakter *escape*. Perhatikan sintaks berikut:

```
1 SELECT
2 RDB$RELATION_NAME
3 FROM RDB$RELATIONS
4 WHERE RDB$RELATION_NAME LIKE '%#_%' ESCAPE '#'
```

2.7 Logika *Boolean*

Subbab ini merupakan kelanjutan dari materi klausa **WHERE** yang telah dijelaskan sebelumnya. Logika *Boolean* yang akan dipelajari adalah **AND**, **OR**, **NOT**, **BETWEEN**, **IN**, dan **IS NULL**.

Perhatikan Tabel(2.3). Berikut adalah contoh penggunaan sintaks dengan logika **AND**:

```
1 SELECT
2 NamaDepan,
3 NilaiUTS
4 FROM Mahasiswa
5 WHERE NilaiUTS >78
6 AND NilaiUTS <80;
```

Berikut adalah contoh untuk logika **OR**:

```
1 SELECT
2 NamaDepan,
3 NilaiUTS,
4 NilaiUAS
5 FROM Mahasiswa
6 WHERE NilaiUTS >79
7 OR NilaiUAS <85;
```

Perhatikan sintaks berikut:

```
1 SELECT
2 NamaDepan,
3 Kegemaran,
4 NilaiUAS
5 FROM Mahasiswa
6 WHERE Kegemaran = 'Olahraga'
7 OR Kegemaran = 'Membaca'
8 AND NilaiUAS ≥ 78;
```

Lalu bandingkan dengan sintaks berikut:

```
1 SELECT
2 NamaDepan,
3 Kegemaran,
4 NilaiUAS
```

```
5 FROM Mahasiswa
6 WHERE (Kegemaran = 'Olahraga'
7 OR Kegemaran = 'Membaca')
8 AND NilaiUAS ≥ 78;
```

Berikut adalah contoh sintaks untuk logika **NOT**:

```
1 SELECT
2 NamaDepan,
3 Kegemaran,
4 NilaiUAS
5 FROM Mahasiswa
6 WHERE NOT Kegemaran = 'Membaca'; /* WHERE Kegemaran <> 'Membaca' */
```

Perhatikan sintaks berikut:

```
1 SELECT
2 NamaDepan,
3 NilaiUTS
4 FROM Mahasiswa
5 WHERE NilaiUTS ≥ 79
6 AND NilaiUTS ≤ 81;
```

Lalu bandingkan dengan sintaks berikut:

```
1 SELECT
2 NamaDepan,
3 NilaiUTS
4 FROM Mahasiswa
5 WHERE NilaiUTS BETWEEN 79 AND 81;
```

dan sintaks berikut:

```
1 SELECT
2 NamaDepan,
3 NilaiUTS
4 FROM Mahasiswa
```

```
5 WHERE NilaiUTS NOT BETWEEN 79 AND 81;
```

Sintaks berikut ini

```
1 SELECT
2 NamaDepan,
3 NilaiUTS
4 FROM Mahasiswa
5 WHERE NilaiUTS = 79
6 AND NilaiUTS = 80;
```

ekivalen dengan sintaks berikut:

```
1 SELECT
2 NamaDepan,
3 NilaiUTS
4 FROM Mahasiswa
5 WHERE NilaiUTS IN(79,80);
```

Bandingkan dengan sintaks berikut:

```
1 SELECT
2 NamaDepan,
3 NilaiUTS
4 FROM Mahasiswa
5 WHERE NilaiUTS NOT IN(79,80);
```

2.8 Logika Kondisi

Perhatikan Tabel (2.8). Kategori F menyatakan buah (*fruit*), S menyatakan bumbu (*spice*), V menyatakan sayur (*vegetable*), dan B menyatakan minuman (*beverage*).

```
1 SELECT
2 CASE CategoryCode
```

```
3 WHEN 'F' THEN 'Fruit'
4 WHEN 'V' THEN 'Vegetable'
5 ELSE 'Other'
6 END AS Category,
7 Description
8 FROM BahanMakanan;
```

Bandingkan dengan sintaks berikut:

```
1 SELECT
2 CASE
3 WHEN CategoryCode = 'F' THEN 'Fruit'
4 WHEN CategoryCode = 'V' THEN 'Vegetable'
5 ELSE 'Other'
6 END AS Category,
7 Description
8 FROM BahanMakanan;
```

Tabel 2.8: BahanMakanan

GroceryID	CategoryCode	Description
1	F	Apple
2	F	Orange
3	S	Mustard
4	V	Carrot
5	B	Water

Tabel 2.9: Amerika

CityID	Country	State	Province	City
1	US	VT		Burlington
2	CA		QU	Montreal
3	US	CO		Denver
4	US	CO		Boulder
5	CA		AB	Edmonton

Perhatikan Tabel (2.9). Misalkan ingin mengurutkan data berdasarkan atribut *Country*, lalu atribut yang menyatakan *State* atau *Province*, dan terakhir berdasarkan *City*. Berikut adalah sintaksnya:

```
SELECT *
FROM Amerika
ORDER BY
Country,
CASE Country
WHEN 'US' THEN State
WHEN 'CA' THEN Province
ELSE State
END,
City;
```

Tabel 2.10: Penghasilan

ID_Karyawan	Jenis_Kelamin	Usia	Pendapatan
1	L	55	80000
2	P	25	65000
3	L	35	40000
4	P	42	90000
5	P	27	25000

Perhatikan Tabel (2.10). Misalkan ingin melihat data karyawan yang tidak sesuai UMR. Jika pria dan minimal 50 tahun, maka penghasilan minimalnya adalah 75000. Jika wanita dan minimal 35 tahun, maka penghasilan minimalnya adalah 60000. Semua orang di regional tersebut harus berpenghasilan minimal 50000.

```
SELECT *
FROM Penghasilan
WHERE Pendapatan <
CASE
WHEN Jenis_Kelamin = 'L' AND Usia ≥ 50 THEN 75000
WHEN Jenis_Kelamin = 'P' AND Usia ≥ 35 THEN 60000
ELSE 50000
END;
```

2.9 Ringkasan Data

Perhatikan Tabel (2.11). Untuk mengeliminasi duplikat, dapat menggunakan kata kunci **DISTINCT**.

```
1 SELECT
2 DISTINCT
3 Artis
4 FROM JudulLagu
5 ORDER BY Artis;
```

Tabel 2.11: JudulLagu

ID_lagu	Artis	Album	Judul
1	The Beatles	Revolver	Yellow Submarine
2	The Beatles	Revolver	Eleanor Rigby
3	The Beatles	Abbey Road	Here Comes the Sun
4	The Rolling Stones	Beggars Banquet	Sympathy for the Devil
5	The Rolling Stones	Let It Bleed	Gimme Shelter
6	Paul McCartney	Ram	Too Many People

Kata kunci **DISTINCT** hanya menampilkan nilai yang unik dalam atribut. Namun Kata kunci **DISTINCT** dapat diikuti dengan atribut lainnya.

```
1 SELECT
2 DISTINCT
3 Artist,
4 Album
5 FROM JudulLagu
6 ORDER BY Artist, Album;
```

Fungsi agregat yang sering dipakai adalah **COUNT**, **SUM**, **AVG**, **MIN**, dan **MAX**. Perhatikan kembali Tabel (2.3) dan gunakan sintaks berikut:

```
1 SELECT
2 MIN(NilaiUTS) AS Minimal_Olahraga,
3 MAX(NilaiUTS) AS Maksimal_Olahraga,
4 SUM(NilaiUTS) AS Total_Olahraga,
```

```
5 AVG(NilaiUTS) AS Rata_rata_Olahraga
6 FROM Mahasiswa
7 WHERE Kegemaran = 'Olahraga';
```

Fungsi **COUNT** digunakan untuk menghitung banyaknya data (baris). Perhatikan sintaks berikut:

```
1 SELECT
2 COUNT(*) AS Banyak_data
3 FROM Mahasiswa
4 WHERE Kegemaran = 'Membaca';
```

Bandingkan dengan sintaks berikut:

```
1 SELECT
2 COUNT(NilaiUTS) AS 'Banyak_data'
3 FROM Mahasiswa
4 WHERE Kegemaran = 'Membaca';
```

Fungsi **GROUP** digunakan untuk memisahkan data menjadi beberapa kelompok. Misalkan ingin mengetahui rata-rata nilai UTS berdasarkan kegemaran. Perhatikan sintaks berikut:

```
1 SELECT
2 Kegemaran AS Jenis_Kegemaran,
3 AVG(NilaiUTS) AS Rata_rata_UTS
4 FROM Mahasiswa
5 GROUP BY Kegemaran
6 ORDER BY Kegemaran;
```

Secara umum, sintaks untuk memisahkan beberapa kolom adalah sebagai berikut:

```
1 SELECT
2 Atribut_1, /*atribut sebenarnya atau fungsi dari atribut*/
3 Atribut_2,
4 ...,
```

```
5 Atribut_n
6 FROM Entitas
7 GROUP BY Atribut_A, Atribut_B, ...
8 ORDER BY Atribut_A, Atribut_B, ... /* boleh urutannya tidak sama ...
      dengan klausa GROUP*/
```

BAB 3

Implementasi Lazarus

Software yang digunakan dalam Bab 3 ini adalah Lazarus. *Software* ini adalah salah satu *open-source software* yang dapat menghubungkan *server* Firebird ke dalam pembuatan aplikasi. Lazarus dapat diunduh di http://www.lazarus-ide.org. Perlu diingat bahwa untuk mengakses *database* Firebird dalam Lazarus, *server* Firebird harus dikoneksikan terlebih dahulu.

Ada beberapa tahapan dalam menggunakan dua *software* tersebut. Pertama, buat aplikasi Lazarus dan persiapkan *database* Firebird yang akan diolah. Kemudian taruh *database* tersebut ke dalam *folder* yang sama dengan aplikasi Lazarus. Lalu pilih komponen ***SQLdb*** untuk aplikasi Lazarus. Perhatikan Gambar (3.1), bahwa *database* yang dapat digunakan selain Firebird adalah Microsoft SQL Server, MySQL, PostgreSQL, Oracle, SQLite, Sybase ASE dan database lain yang menggunakan ODBC.

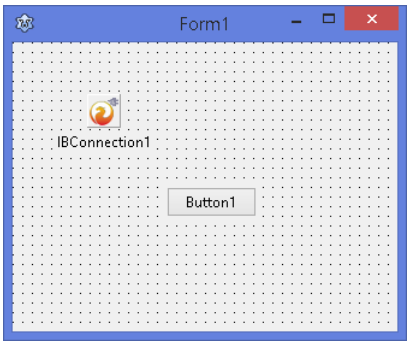
Gambar 3.1: Komponen-komponen ***SQLdb***

Database yang digunakan dalam materi ini adalah Firebird, sehingga komponen yang dipilih adalah ***IBConnection***. Letakkan komponen tersebut pada **form**, dan tambahkan ***Button*** (dari komponen *Standard*) sehingga tampilan **form** seperti pada Gambar (3.2). Pada komponen ***Button***, tambahkan ***OnClick*** *event* lalu beri sintaks berikut:

```
1 IBConnection1.DatabaseName:='mydatabase.FDB';
2 IBConnection1.Connected:=True;
3 ShowMessage('Selamat! Database terhubung.');
```

Jangan lupa, atur properti untuk komponen ***IBConnection*** sebagai berikut:

1. *Hostname=localhost*
2. *UserName=SYSDBA*
3. *Password=masterkey.*



Gambar 3.2: Komponen ***IBConnection*** pada **form**

3.1 Komponen DBGrid

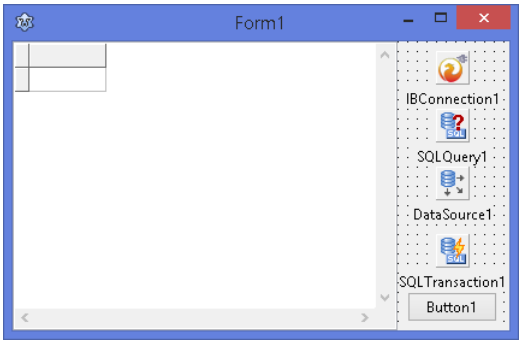
Komponen ***SQLTransaction*** dan ***SQLQuery*** dapat digunakan untuk semua *database* yang didukung oleh ***SQLdb***. Untuk menampilkan *database*, dapat menggunakan komponen ***DBGrid*** yang tersedia dalam *Data Controls*. Lalu untuk menghubungkan komponen ***DBGrid***, diperlukan komponen ***Datasource*** yang tersedia dalam *Data Access*.

Selanjutnya adalah menghubungkan komponen ***DBConnection***, ***SQLTransaction***, ***SQLQuery***, ***DBGrid***, dan ***Datasource***. Pada properti ***IBConnection***, ubah *Transaction* menjadi 'SQLTransaction1'. Lalu ubah properti *Database* pada ***SQLQuery*** menjadi 'IBConnection1' dan properti *Transaction* menjadi

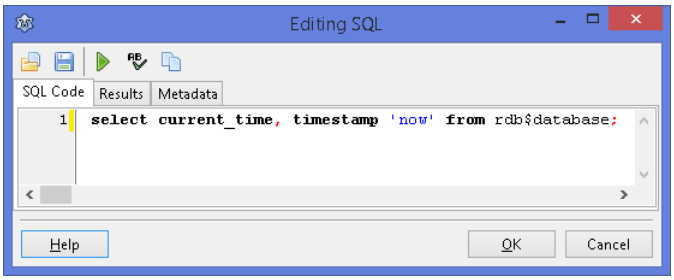
'SQLTransaction1'. Kemudian ubah pula properti *Dataset* pada ***Datasource*** menjadi 'SQLQuery1'. Terakhir, properti *Datasource* pada ***DBGrid*** diubah menjadi 'DataSource1'.

Setelah komponen-komponen terhubung, lalu buat 1 buah ***Button*** dan atur seperti Gambar (3.3). Selanjutnya tambahkan ***OnClick*** *Event* pada ***Button*** dengan sintaks berikut:

```
1 SQLQuery1.Close;
2 SQLQuery1.SQL.Text:= 'select * from ENTITAS';
3 IBConnection1.Connected:= True;
4 SQLTransaction1.Active:= True;
5 SQLQuery1.Open;
6 DBGrid1.AutoFillColumns:=True; //opsional
```



Gambar 3.3: Komponen-komponen yang terhubung ke ***DBGrid***



Gambar 3.4: Jendela "*Editing* SQL"

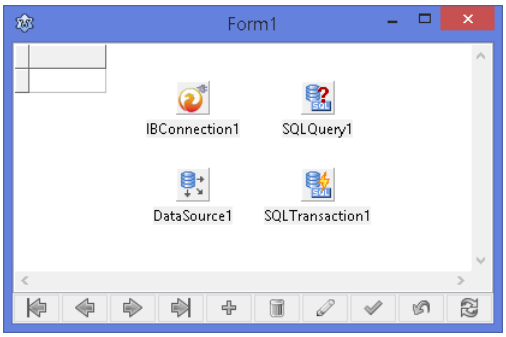
Untuk mengetahui bagaimana komponen ***SQLQuery*** bekerja, tekan edit pada properti *SQL* dalam ***SQLQuery***. Maka akan muncul jendela "*Editing* SQL", lalu tambahkan sintaks seperti pada Gambar (3.4). Untuk mengetahui hasilnya, tekan tanda segitiga hijau (*Run* SQL *code*).

3.2 Komponen DBNavigator dan DBEdit

Selain ***DBGrid***, untuk menampilkan *database* juga dapat menggunakan komponen ***DBNavigator*** dan ***DBEdit***. Kedua komponen ini juga dapat dipadukan dengan komponen ***DBGrid***. Komponen ***DBNavigator*** dan ***DBEdit*** tersedia dalam *Data Controls*.

Gambar 3.5: Komponen ***DBNavigator***

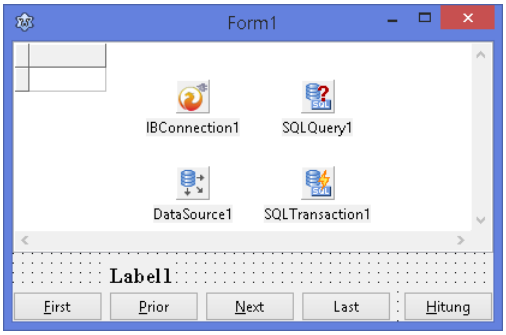
DBNavigator digunakan untuk menavigasi melalui *dataset* dalam merespon *query* yang dikirim ke *database*. Perhatikan Gambar (3.5), bahwa komponen ***DBNavigator*** terdiri dari beberapa tombol yakni ***First***, ***Prior***, ***Next***, ***Last***, ***Insert***, ***Delete***, ***Edit***, ***Post***, ***Cancel***, dan ***Refresh***. Namun pada saat mendesain, tombol-tombol tersebut dapat ditampilkan atau disembunyikan dengan menyesuaikan properti *VisibleButtons*. Untuk menghubungkan komponen ***DBNavigator*** dengan ***DBGrid***, maka pada properti *Dataset* pada ***Datasource*** menjadi 'SQLQuery1' dan properti *Datasource* pada ***DBNavigator*** diubah menjadi 'DataSource1'. Selanjutnya adalah menyesuaikan komponen-komponen tersebut seperti pada Gambar (3.6).



Gambar 3.6: Contoh desain **form**

Yang menarik dari pembahasan ***DBNavigator*** ini adalah perhitungan banyaknya data. Untuk menghitung banyaknya data dalam *dataset*, gunakan fasilitas *RecordCount* yang disediakan oleh ***SQLQuery***. Nilai yang dihasilkan oleh *RecordCount* menunjukkan banyaknya data yang dimuat dari *server*. Untuk alasan performa, komponen ***SQLdb*** tidak membaca semua data ketika membuka ***SQLQuery***.

Dalam hal ini, hanya 10 data saja yang tersedia. Sehingga untuk mengetahui banyaknya data dalam *server*, gunakan navigasi *Last* lalu panggil *RecordCount*.



Gambar 3.7: Contoh desain **form**

Untuk lebih jelasnya, ganti komponen ***DBNavigator*** dengan 4 buah ***Button*** dengan *Caption* masing-masing *First*, *Prior*, *Next*, dan *Last*. Lalu tambahkan 1 ***Label*** dan 1 ***Button*** tambahan (yang digunakan untuk menghitung banyak data). Kemudian tambahkan beberapa ***procedure*** berikut.

```
1  procedure TForm1.Button1Click(Sender: TObject);
2  begin
3  SQLQuery1.First;
4  end;
5
6  procedure TForm1.Button2Click(Sender: TObject);
7  begin
8  SQLQuery1.Prior;
9  end;
10
11 procedure TForm1.Button3Click(Sender: TObject);
12 begin
13 SQLQuery1.Next;
14 end;
15
16 procedure TForm1.Button4Click(Sender: TObject);
17 begin
18 SQLQuery1.Last;
19 end;
20
21 procedure TForm1.Button5Click(Sender: TObject);
22 var n:integer;
```

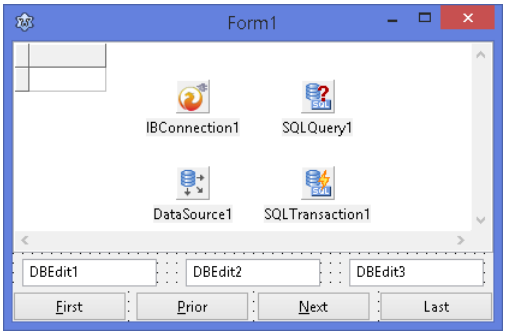
```
23 begin
24 n:=SQLQuery1.RecordCount;
25 Label1.Caption:='Banyaknya data = '+IntToStr(n);
26 end;
```

Selanjutnya sesuaikan komponen-komponen tersebut dengan mengikuti Gambar (3.7), lalu perhatikan hasil *output*-nya.

Komponen selanjutnya adalah ***DBEdit***. Komponen ***DBEdit*** menunjukkan satu entri *database* dalam kotak *edit*. Oleh karena itu, komponen ini cocok sekali ketika dipadukan dengan komponen ***DBNavigator***. Untuk lebih jelasnya, isikan properti *SQL* pada komponen ***SQLQuery*** dengan sintaks berikut.

```
1 select * from entitas;
```

Lalu tambahkan 3 ***DBEdit***. Kemudian ubah properti *Datasource* pada masing-masing ***DBEdit*** menjadi 'DataSource1'. Selanjutnya pilih *DataField* sesuai atribut yang ingin ditampilkan. Terakhir, atur komponen-komponen tersebut seperti pada Gambar (3.8).



Gambar 3.8: Contoh desain **form**

3.3 Memanipulasi *Database*

Perintah yang termasuk dalam memanipulasi data meliputi menambah, menghapus, dan memperbaharui. Berikut adalah contoh sintaks untuk menambahkan data.

```
1 SQLQuery1.SQL.Text:='insert into entitas values (''teks'',1234)';
```

Cara lain untuk menambah data adalah menggunakan parameter dalam *query*.

```
1 SQLQuery1.SQL.Text:='insert into entitas values (:a,:b)';
2 SQLQuery1.Params.ParamByName('a').AsString:='teks';
3 SQLQuery1.Params.ParamByName('b').AsInteger:=1234;
```

Hal ini sangat berguna ketika ingin menambahkan data yang berasal dari komponen Lazarus yang lain. Perhatikan sintaks berikut.

```
1 SQLQuery1.SQL.Text:='insert into entitas values (:a,:b)';
2 SQLQuery1.Params.ParamByName('a').AsString:=Edit1.Text;
3 SQLQuery1.Params.ParamByName('b').AsInteger:=StrToInt(Edit2.Text);
```

Seperti penjelasan pada materi sebelumnya, bahwa jika ingin memanipulasi data (menambah, menghapus, dan memperbaharui) maka dalam bahasa SQL menggunakan perintah **COMMIT**. Jika ingin melakukan perintah ini dalam Lazarus, maka gunakan sintaks berikut setelah data dimanipulasi.

```
1 SQLQuery1.ExecSQL;
2 SQLTransaction1.Commit;
```

3.4 Memanfaatkan StringGrid

Berbeda dengan ***DBGrid***, komponen ***StringGrid*** merupakan komponen yang dapat digunakan untuk menyajikan (memodifikasi) *database* secara langsung sesuai keinginan Pengguna. Beberapa properti yang sering digunakan adalah *ColCount*, *RowCount*, dan *Cells*[kolom, baris]. Materi ini menggunakan *database* yang disediakan oleh Firebird, yaitu "employee.fdb". Koneksikan *database* tersebut dengan Lazarus. Lalu tambahkan 1 ***Button*** dengan sintaks berikut:

```
1 procedure TForm1.Button1Click(Sender: TObject);
```

```
2  var
3    i,m,n:integer;
4    gl,gb:Double;
5  begin
6    SQLQuery1.Close;
7    SQLQuery1.SQL.Text:='select old_salary as gaji_lama, ...
         new_salary as gaji_baru from salary_history';
8    SQLQuery1.Open;
9    SQLQuery1.Last;
10
11   m:=SQLQuery1.FieldCount;
12   n:=SQLQuery1.RecordCount;
13   StringGrid1.ColCount:=m+1;
14   StringGrid1.RowCount:=n+1;
15   StringGrid1.AutoFillColumns:=True;
16
17   StringGrid1.Cells[0,0]:='No.';
18   StringGrid1.Cells[1,0]:='Gaji Sebelum';
19   StringGrid1.Cells[2,0]:='Gaji Sesudah';
20
21   SQLQuery1.First;
22
23   for i:=1 to n do
24   begin
25     gl:=SQLQuery1.FieldByName('gaji_lama').AsFloat;
26     gb:=SQLQuery1.FieldByName('gaji_baru').AsFloat;
27
28     StringGrid1.Cells[0,i] := IntToStr(i);
29     StringGrid1.Cells[1,i] := FloatToStr(gl);
30     StringGrid1.Cells[2,i] := FloatToStr(gb);
31     SQLQuery1.Next;
32   end;
33 end;
```

3.5 Menggunakan LazReport

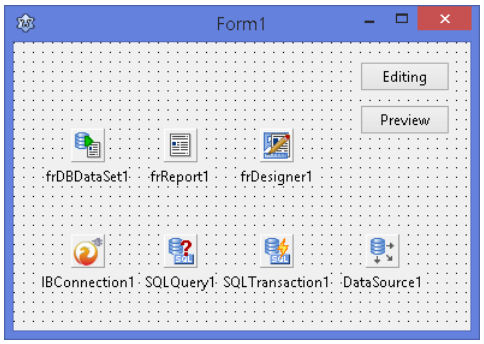
Komponen ***LazReport*** dapat melihat dan mencetak data yang disusun dengan baik. Namun komponen ini biasanya belum terinstall, sehingga harus diinstall terlebih dahulu. Dari menu *Package*, klik submenu *Install/Uninstall Packages*. Lalu

pada editor *Available for installation*, ketik lazreport dan klik *Install selection*. Jika berhasil, maka komponen ***LazReport*** akan muncul dalam beberapa menit.

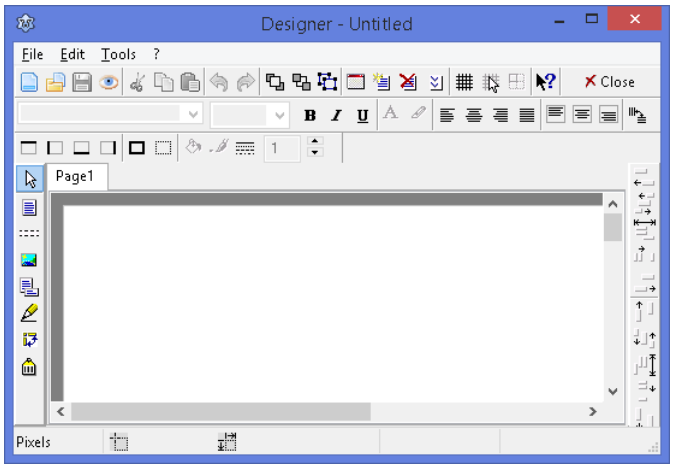
Komponen utama yang akan digunakan adalah ***frDBDataSet***, ***frReport*** dan ***frDesigner***. Letakkan ketiga komponen tersebut pada **form**, lalu tambahkan beberapa komponen pendukung sehingga **form** akan tampak seperti pada Gambar (3.9). Materi ini menggunakan *database* yang disediakan oleh Firebird, yaitu "employee.fdb". Koneksikan *database* tersebut dengan Lazarus. Lalu pada komponen **SQLQuery**, tambahkan sintaks berikut untuk properti *SQL*:

```
1 select * from employee;
```

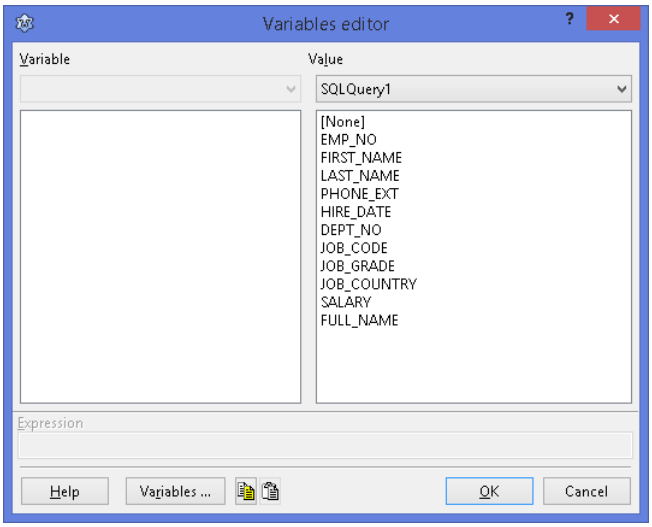
Kemudian properti *Datasource* pada **frDBDataSet** diubah menjadi 'DataSource1'.



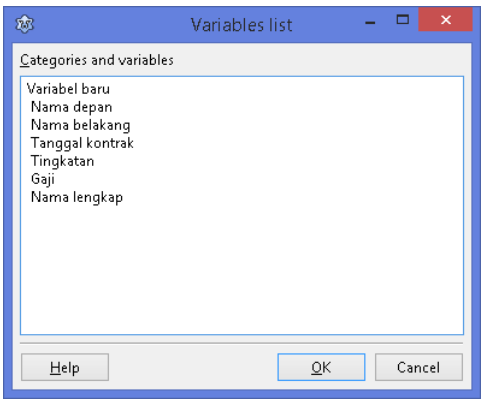
Gambar 3.9: Contoh desain **form**



Gambar 3.10: *Report designer*

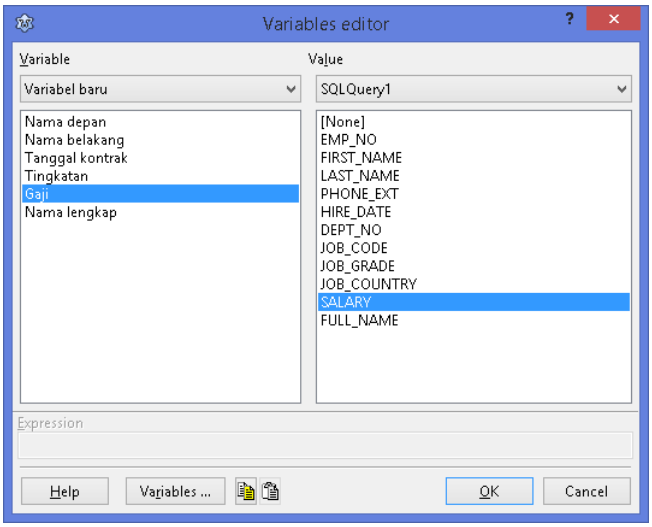


Gambar 3.11: *Variables editor*

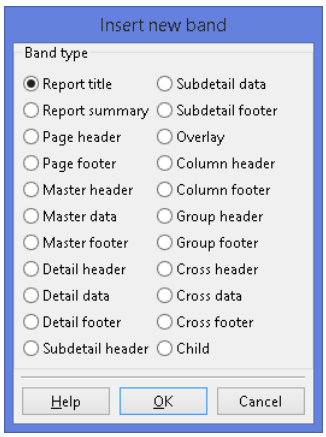


Gambar 3.12: *Variables list*

Selanjutnya adalah membuat tampilan *report*. *Double-click* pada komponen ***frReport*** (atau klik kanan, lalu pilih *Design Report*) sehingga muncul *report designer* seperti pada Gambar (3.10). Selanjutnya adalah menghubungkan nama-nama variabel ke *database fields* atau nilai-nilai sistem. Dari menu *File*, klik *variables list* sehingga akan tampak seperti pada Gambar (3.11). Klik tombol *Variables* untuk mendefinisikan beberapa variabel. Tulis beberapa variabel seperti pada Gambar (3.12). "Variabel baru" menyatakan suatu kategori, sedangkan entri-entri yang berada dibawahnya menyatakan variabel. Pendefinisian variabel diawali dengan 1 spasi. Selanjutnya, setiap variabel harus dihubungkan dengan suatu *field*. Pilih variabel, lalu pilih *field* yang sesuai, maka akan tampak seperti pada Gambar (3.13). Lakukan hal yang sama untuk semua variabel.

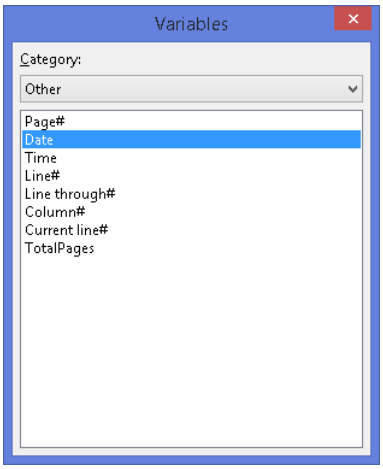


Gambar 3.13: Contoh pencocokan variabel dan *field*

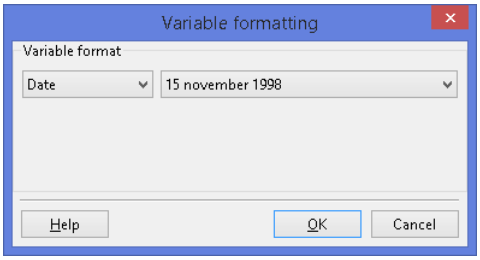


Gambar 3.14: Tipe-tipe *band*

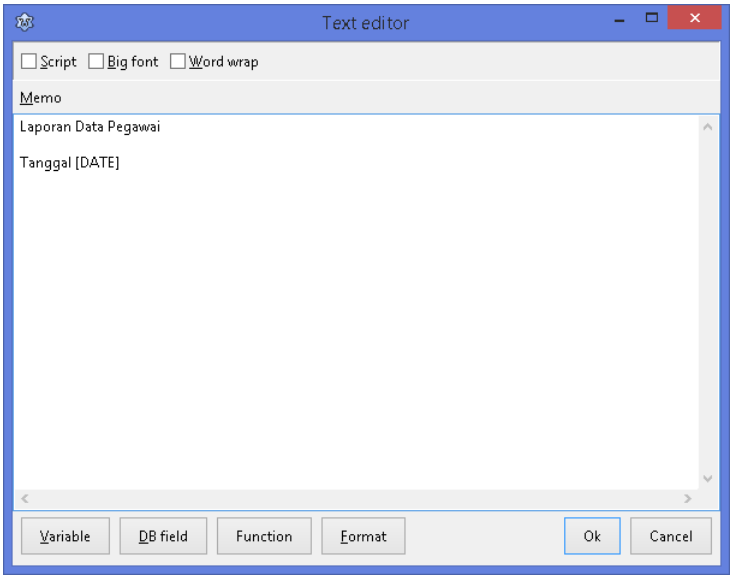
Selanjutnya adalah menggunakan objek-objek yang ada pada *object bar*. Posisi *object bar* terletak pada sisi kiri dari *report designer*. Objek-objek tersebut terdiri dari ***pointer***, ***rectangle object***, ***band***, ***picture object***, ***subreport***, ***draw lines***, ***TlrCrossView*** dan ***barcode object***. Klik objek ***band***, lalu klik juga pada *report surface*. Setelah itu muncul beberapa tipe *band* seperti pada Gambar (3.14).



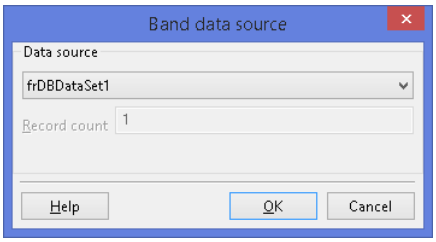
Gambar 3.15: Cara memilih tanggal sekarang



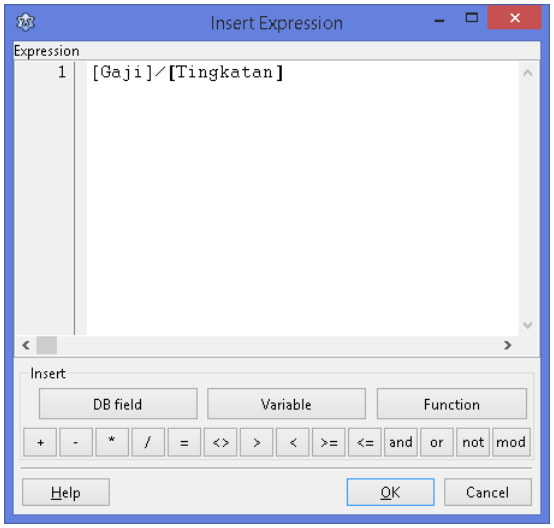
Gambar 3.16: Jendela *variables formatting*



Gambar 3.17: Contoh desain *text editor*



Gambar 3.18: Pemilihan *data source* untuk *master data band*



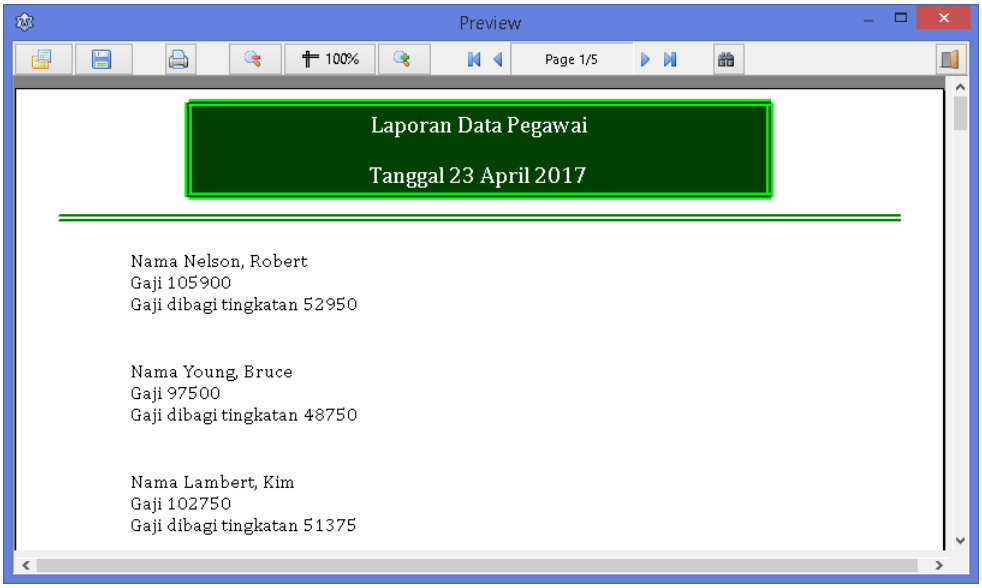
Gambar 3.19: Menambahkan ekspresi pada *text editor*

Tipe *band* pertama yang akan dipelajari adalah *report title*. Pilih tipe band ini, lalu letakkan ***rectangle object*** di atasnya. Setelah itu, akan muncul *text editor*. Untuk menambahkan tanggal saat ini, pilih *variable* lalu pilih kategori *other* kemudian *Date* seperti pada Gambar (3.15). Untuk mengatur hasil tampilan, pilih *format* pada *text editor*. Misalkan ingin mengatur tampilan tanggal seperti pada Gambar (3.16). Selanjutnya adalah menyesuaikan dengan Gambar (3.17).

Tipe *band* selanjutnya adalah *master data*. Pilih ***frDBDataSet*** sebagai *data source* seperti pada Gambar (3.18). Tipe *band* ini akan mengulang setiap *record* dalam *data source*. Untuk menambahkan beberapa *field*, tambahkan ***rectangle object*** di atasnya. Tambahkan beberapa variabel dengan kategori "Variabel baru". Untuk perhitungan numerik, dapat menggunakan *function* seperti pada Gambar (3.19).

Setelah membuat desain dasar untuk *report*, saatnya mengetahui hasil tampilan. Dari menu *File*, pilih *preview* (tahan Ctrl+P dari *keyboard*). Contoh hasil tampilan *report* dapat dilihat pada Gambar (3.20). Tutup dan simpan desain *report* dengan nama "latihanlazreport.lrf". Langkah selanjutnya adalah menambahkan beberapa sintaks berikut pada **form**.

```
1  procedure TForm1.TombolEditClick(Sender: TObject);
2  begin
3  frReport1.LoadFromFile('latihanlazreport.lrf');
4  frReport1.DesignReport;
5  end;
6
7  procedure TForm1.TombolPreviewClick(Sender: TObject);
8  begin
9  frReport1.LoadFromFile('latihanlazreport.lrf');
10 frReport1.ShowReport;
11 end;
```



Gambar 3.20: Contoh tampilan *report*

Report yang telah dibuat dapat dicetak dengan menambahkan "Printer4Lazarus" *package*. Tambahkan 1 ***Button*** dan 1 ***PrintDialog***. Pada komponen ***Button***, tambahkan ***OnClick*** *event* lalu beri sintaks berikut:

```
1  procedure TForm1.TombolPrintClick(Sender: TObject);
2  var
3    FromPage, ToPage, NumberCopies: Integer;
4    ind: Integer;
5    Collap: Boolean;
6  begin
7    // Load report definition from application directory
8    AppDirectory:=ExtractFilePath(ParamStr(0));
9    frReport1.LoadFromFile('latihanlazreport.lrf');
10   // Need to keep track of which printer was originally selected ...
        to check for user changes
11   ind:= Printer.PrinterIndex;
12   // Prepare the report and just stop if we hit an error as ...
        continuing makes no sense
13   if not frReport1.PrepareReport then Exit;
14   // Set up dialog with some sensible defaults which user can change
15   with PrintDialog1 do
16   begin
17     Options:=[poPageNums ]; // allows selecting pages/page numbers
18     Copies:=1;
19     Collate:=true; // ordened copies
20     FromPage:=1; // start page
21     ToPage:=frReport1.EMFPages.Count; // last page
22     MaxPage:=frReport1.EMFPages.Count; // maximum allowed number ...
          of pages
23     if Execute then // show dialog; if succesful, process user ...
          feedback
24     begin
25      if (Printer.PrinterIndex <> ind ) // verify if selected ...
           printer has changed
26       or frReport1.CanRebuild // ... only makes sense if we can ...
           reformat the report
27       or frReport1.ChangePrinter(ind, Printer.PrinterIndex) ...
           //... then change printer
28       then
29       frReport1.PrepareReport //... and reformat for new printer
30      else
31       exit; // we could not honour the printer change
32
33      if PrintDialog1.PrintRange = prPageNums then // user made ...
           page range selection
```

```
34        begin
35         FromPage:=PrintDialog1.FromPage; // first page
36         ToPage:=PrintDialog1.ToPage;  // last page
37        end;
38        NumberCopies:=PrintDialog1.Copies; // number of copies
39        // Print the report using the supplied pages & copies
40        frReport1.PrintPreparedReport( ...
             inttostr(FromPage)+'-'+inttostr(ToPage), NumberCopies);
41       end;
42     end;
43  end;
```

Daftar Pustaka

[1] Borrie, H. (2004). *The Firebird Book: A Reference for Database Developers*. New York: Apress.

[2] Fathansyah. (2012). *Basis Data*, Edisi Revisi. Bandung: Informatika.

[3] Filippov, D., Karpeykin, A., Kovyazin, A., Kuzmenko, D., Simonov, D., Vinkenoog, P., Yemanov, D. (2016). *Firebird 2.5 Language Reference*. http://www.firebirdtest.com/

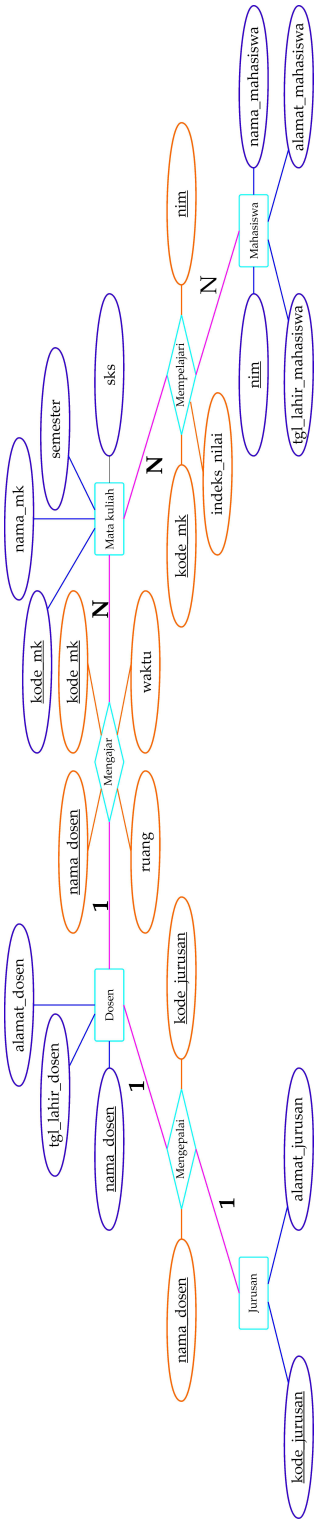
[4] Rockoff, L. (2017). *The Language of SQL*, Second Edition. New York: Addison-Wesley.

[5] Silberchatz, A., Korth, Henry, F., Sudarshan, S. (2011). *Database System Concepts*, Sixth Edition. New York: Mc Graw Hill.

[6] http://wiki.lazarus.freepascal.org.

Lampiran

Lampiran 1


nama_mahasiswa
alamat_mahasiswa
nim
Mahasiswa
N
Mempelajari
sks
semester
nama_mk
Mata kuliah
N
indeks_nilai
kode_mk
nim
tgl_lahir_mahasiswa
kode_mk
kode_mk
N
waktu
Mengajar
nama_dosen
ruang
1
alamat_dosen
Dosen
kode_jurusan
tgl_lahir_dosen
nama_dosen
1
Mengepalai
1
alamat_jurusan
nama_dosen
Jurusan
kode_jurusan

Lampiran 2

Tipe Data dalam SQL

No.	Nama	Ukuran	Batas & ketelitian	Keterangan
1	BIGINT	64 *bits*	Dari -2^{63} sampai $(2^{63}-1)$	Tipe data untuk bilangan bulat dengan rentang yang lebih lebar dari INT
2	BLOB	Bervariasi	Ukuran segmen BLOB dibatasi hingga 64K dan ukuran maksimum file BLOB adalah 4 GB	Suatu tipe data dalam ukuran yang dinamik yang digunakan untuk menyimpan data yang cukup besar seperti grafik, teks, dan suara dalam bentuk digital
3	CHAR(n), Character(n)	Karena terdiri dari n karakter, maka ukuran tipe data ini tergantung pada banyaknya *bytes* dalam setiap karakter	dari 1 sampai 32767 *bytes*	Tipe data karakter yang panjangnya tetap. Artinya jika telah mendeklarasikan suatu variabel bertipe Char(8), maka alokasi ukuran penyimpanan adalah sebesar lima karakter
4	DATE	32 *bits*	Dari 01.01.0001 masehi sampai 31.12.9999 masehi	Variabel ini hanya mengenal tanggal, tidak ada unsur waktu
5	DECIMAL (precision, scale)	Bervariasi (16, 32 atau 64 *bits*)	Precision = dari 1 sampai 18, menyatakan total digit dari bilangan yang akan disimpan; scale = dari 0 sampai 18, menyatakan banyaknya digit di belakang tanda desimal	Suatu bilangan dengan beberapa digit setelah titik desimal. Banyaknya scale harus kurang dari atau sama dengan precision. Variabel dengan tipe DECIMAL(10,3) akan mengikuti format: ppppppp.sss, contohnya 1234567.890
6	DOUBLE PRECISION	64 *bits*	Dari $2.225 * 10^{-308}$ sampai $1.797 * 10^{308}$	Double-precision IEEE. Tipe data ini disimpan dengan ketepatan perkiraan 15 digit (tergantung pada *platform*)

Lampiran 2

Tipe Data dalam SQL (Lanjutan)

No.	Nama	Ukuran	Batas & ketelitian	Keterangan
7	FLOAT	32 *bits*	Dari $1.175 * 10^{-38}$ sampai $3.402 * 10^{38}$	Single-precision IEEE. Tipe data ini memiliki perkiraan presisi 7 digit setelah titik desimal
8	INTEGER, INT	32 *bits*	Dari -2147483648 sampai 2147483647	Tipe data untuk bilangan bulat yang sering digunakan. Nama pendek untuk tipe data ini adalah INT
9	NUMERIC (precision, scale)	Bervariasi (16, 32 atau 64 *bits*)	Precision = dari 1 sampai 18, menyatakan total digit dari bilangan yang akan disimpan; scale = dari 0 sampai 18, menyatakan banyaknya digit di belakang tanda desimal.	Suatu bilangan dengan beberapa digit setelah titik desimal. Banyaknya scale harus kurang dari atau sama dengan precision. Variabel dengan tipe NUMERIC(10,3) akan mengikuti format: ppppppp.sss, contohnya 1234567.890
10	SMALLINT	16 *bits*	Dari -32768 sampai 32767	Tipe data untuk bilangan bulat dengan rentang yang lebih sempit dari INT
11	TIME	32 *bits*	Dari 00:00:00.0000 sampai 23:59:59.9999	ISC_TIME, yang menunjukkan waktu. Tipe data ini tidak dapat digunakan untuk menyimpan interval waktu. Dapat menggunakan fungsi *extract*, seperti EXTRACT (HOUR FROM DATE_FIELD), EXTRACT (MINUTE FROM DATE_FIELD), EXTRACT (SECOND FROM DATE_FIELD)
12	TIMESTAMP	64 *bits*	Dari tanggal 01.01.0001 sampai 31.12.9999	Menghasilkan tanggal dan waktu. Dapat juga menggunakan fungsi EXTRACT

Lampiran 2

Tipe Data dalam SQL (Lanjutan)

No.	Nama	Ukuran	Batas & ketelitian	Keterangan
13	VARCHAR(n), Char Varying, Character Varying	Karena terdiri dari n karakter, maka ukurannya bergantung pada banyaknya *bytes* dalam setiap karakter	Dari 1 sampai 32765 *bytes*	Variabel bertipe string. Ukuran total karakter dalam byte tidak boleh lebih besar dari (32KB-3), dengan mempertimbangkan *encoding*. Parameter n wajib diisi