Normalisasi Database
Menggunakan Metode Logika Sederhana

Adhi Hargo
cadmus_sw at yahoo.com

1. Pendahuluan

Normalisasi database biasanya jarang dilakukan dalam database skala kecil, dan dianggap tidak diperlukan pada penggunaan personal. Namun seiring dengan berkembangnya informasi yang dikandung dalam sebuah database, proses normalisasi akan sangat membantu dalam menghemat ruang yang digunakan oleh setiap tabel di dalamnya, sekaligus mempercepat proses permintaan data. Berikut ini dipaparkan metodologi logis sederhana untuk menormalkan model data dalam sebuah database, diiringi contoh pembuatan database untuk tugas-tugas matakuliah dalam sebuah fakultas (fiktif) dengan atribut yang disederhanakan.

Proses normalisasi model data dapat diringkas sebagai berikut:

1. Temukan entitas-entitas utama dalam model data.
2. Temukan hubungan antara setiap entitas.
3. Tentukan atribut yang dimiliki masing-masing entitas.

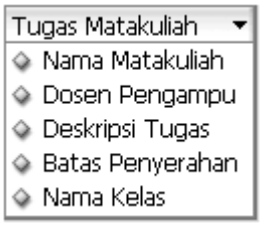
Normalisasi model data dilakukan dengan mengikuti langkah-langkah sederhana, mengubahnya agar memenuhi apa yang disebut sebagai bentuk normal pertama, kedua, lalu ketiga secara berturutan.

2. Langkah-Langkah Normalisasi

2.1. Bentuk Normal Pertama (1NF)

Sebuah model data dikatakan memenuhi bentuk normal pertama apabila setiap atribut yang dimilikinya memiliki satu dan hanya satu nilai. Apabila ada atribut yang memiliki nilai lebih dari satu, atribut tersebut adalah kandidat untuk menjadi entitas tersendiri.

Entitas utama untuk database tugas matakuliah tentu saja Tugas Matakuliah. Sebagian atribut yang dimiliki entitas ini tertera dalam Gambar 1.



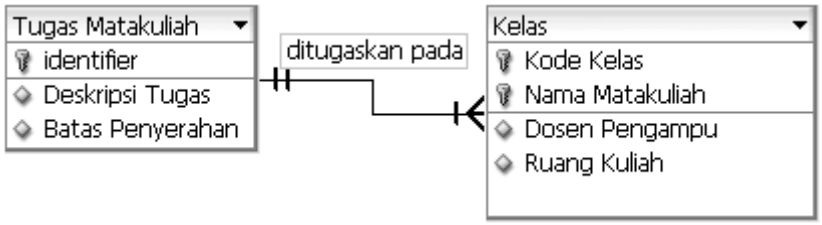
Gambar 1: Entitas pertama dalam contoh model data untuk database tugas matakuliah.

Atribut Nama Kelas mencantumkan kelas-kelas di mana tugas tersebut berlaku. Apabila pendaftar untuk sebuah matakuliah melebihi kapasitas ruangan yang dimiliki fakultas, kebijakan yang umum diambil Kepala Program Studi adalah membagi kegiatan perkuliahan untuk matakuliah tersebut menjadi beberapa kelas. Karenanya atribut ini rentan memiliki nilai jamak, dan lebih sesuai menjadi entitas baru atau atribut dari entitas lain. Untuk sementara kita membuat entitas baru, Kelas, dimana sebagian atributnya berasal dari Tugas Matakuliah yang secara logis lebih sesuai menjadi atribut entitas ini. Sementara itu, hampir semua atribut entitas Tugas Matakuliah selain Nama Kelas memiliki nilai tunggal (dengan asumsi setiap matakuliah diampu oleh satu dosen saja).

2.1.a Relasi Antar-Entitas dan Identifier

Masalah yang kita hadapi sekarang adalah menghubungkan Tugas Matakuliah dengan Kelas. Satu tugas dapat diberikan pada beberapa kelas yang berbeda; dalam terminologi pemodelan data, ini berarti antara entitas Tugas Matakuliah dan entitas Kelas terdapat **relasi 1:N** (atau 1-N) untuk nilai N lebih dari satu. Cara paling intuitif untuk menghubungkan kedua entitas tersebut adalah menyertakan **identitas** satu entitas sebagai atribut entitas lain. Identitas sebuah entitas haruslah unik untuk menghindarkan ambiguitas saat akan merujuk pada satu objek khusus dari entitas tersebut. Entitas Tugas Matakuliah akan menggunakan pengidentifikasi arbitrer berupa angka yang berbeda antara satu objek Tugas Matakuliah dengan objek Tugas Matakuliah lain. Entitas Kelas dapat diidentifikasi dengan matakuliah dan kode kelas yang bersangkutan, sehingga kita cukup menambahkan atribut pengidentifikasi (*identifier*) dalam kedua entitas. Entitas ini beserta semua atribut baru dan hubungannya dengan Tugas Matakuliah diperlihatkan dalam Gambar 2, dengan menggunakan notasi relasi *crows foot* (dengan simbol “kaki gagak”

menunjuk pada entitas jamak).



Gambar 2: Hubungan antara Tugas Matakuliah dan entitas baru, Kelas.

Sejauh ini tidak ada atribut entitas yang memiliki nilai lebih dari satu, sehingga rasanya cukup aman mengatakan bahwa model ini memenuhi bentuk normal pertama.

2.2. Bentuk Normal Kedua (2NF)

Sebuah model data dikatakan memenuhi bentuk normal kedua apabila ia memenuhi bentuk normal pertama dan setiap atribut non-*identifier* sebuah entitas bergantung sepenuhnya hanya pada semua *identifier* entitas tersebut.

Apabila kita perhatikan kembali model data yang telah kita hasilkan di atas, segera terlihat bahwa atribut dari entitas Kelas tidak sepenuhnya bergantung pada identitas unik Kelas tersebut. Seorang dosen akan tetap ada meskipun kelas matakuliah yang ia ampu sudah tidak ada lagi. Dalam hal ini, dosen adalah entitas tersendiri (yang nantinya dapat dilekatkan pada entitas Fakultas atau Universitas bilamana kedua entitas tersebut dirasa perlu ada, tergantung pada kebutuhan pemodelan data kita).

2.2.a Sekali Lagi, Tentang Identifier

Dalam dunia nyata, anggapan yang umum adalah seseorang (“individu”) dapat diidentifikasi secara unik dengan namanya. Tentu saja anggapan ini tidak sepenuhnya benar, karena bisa saja sebuah nama (bahkan satu rangkaian nama lengkap) dimiliki oleh lebih dari satu orang; pemodelan data yang melibatkan informasi tentang individu jarang menggunakan nama individu tersebut sebagai satu-satunya pengidentifikasi. Implementasi RDBMS tertentu juga akan lebih cepat memproses *query* atas suatu tabel apabila tabel tersebut diindeks oleh nilai integer unik daripada bila menggunakan indeks karakter (rangkaian karakter masih harus diumpankan ke fungsi *hash* agar dapat digunakan sebagai indeks tabel, sementara untuk integer unik tidak harus).

Karena beberapa alasan tersebut, entitas Dosen pada model data kita akan menggunakan pengidentifikasi arbitrer berupa Nomor Induk Pegawai sebagaimana diperlihatkan dalam Gambar 3. Dalam notasi *crows foot*, relasi non-*identifying* digambarkan dengan garis putus-putus atau tersamar.

Gambar 3: Ketiga entitas utama dalam model data dan hubungan antar masing-masing entitas.

Setelah atribut-atribut dari semua entitas dalam sebuah model data hanya bergantung pada seluruh pengidentifikasi entitas yang memilikinya, model data tersebut dikatakan memenuhi bentuk normal kedua.

2.3. Bentuk Normal Ketiga (3NF)

Sebuah model data dikatakan memenuhi bentuk normal ketiga apabila ia memenuhi bentuk normal kedua dan tidak ada satupun atribut non-*identifying* (bukan pengidentifikasi unik) yang bergantung pada atribut non-*identifying* lain. Apabila ada, pisahkan salah satu atribut tersebut menjadi entitas baru, dan atribut yang bergantung padanya menjadi atribut entitas baru tersebut.

Dalam model data sederhana yang kita gunakan di sini, tidak ada satupun atribut non-*identifying* (seperti Deskripsi Tugas Matakuliah, atau Nama Dosen) yang bergantung pada atribut non-*identifying* lain. Namun demi adanya contoh, kita misalkan entitas Dosen memiliki atribut informasi Alamat Rumah dan Nomor Telepon Rumah. Keduanya tidak dapat secara unik mengidentifikasi objek tertentu dari entitas Dosen, namun keduanya saling bergantung. Sebagaimana dalam dua langkah normalisasi sebelumnya, jenis kebergantungan seperti ini dapat dihilangkan dengan membuat entitas baru lagi (yang tidak akan diciptakan karena tiga entitas sudah cukup banyak untuk satu artikel).

Model terakhir yang kita dapat ini telah memenuhi bentuk normal ketiga (*third normal form*) dan siap dikonversi menjadi tabel. Namun sebelumnya, kita perlu membahas berbagai jenis relasi yang kerap ditemui dalam pemodelan data, termasuk yang kita temui dalam contoh model data kali ini.

3. Jenis-jenis Relasi Antar-Entitas

1. Relasi 1-1. Relasi ini jarang ditemui dalam model data yang benar, sehingga saat Anda menemukannya, kemungkinan besar hal itu berarti masih ada yang belum sempurna dari model data Anda; relasi 1-1 sering berarti kedua entitas tersebut sebenarnya adalah kesatuan, satu entitas tunggal. Kemungkinan lain adalah relasi 1-1 ini adalah relasi turunan atau relasi non-*identifying* (identitas unik satu entitas tidak bergantung pada identitas unik entitas lain) namun jenis relasi kedua ini jarang ditemui.

2. Relasi 1-N. Relasi ini yang paling umum ditemui dalam model data.

3. Relasi M-N. Relasi ini juga sering ditemui dalam model data, dan sering pula dapat dinormalkan lebih jauh lagi. Langkah yang dapat ditempuh untuk menormalkan relasi M-N:

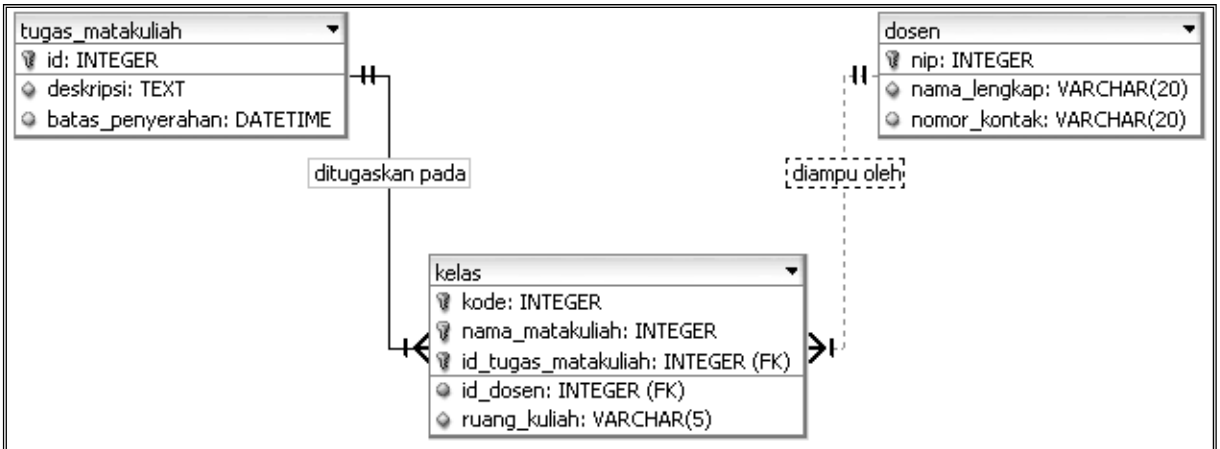
    a. Buat sebuah entitas baru sebagai penghubung antara kedua entitas dengan relasi M-N tersebut. Entitas penghubung ini akan memiliki hubungan 1-M dengan masing-masing entitas awal. *Identifier* entitas penghubung dapat dibuat tersendiri, atau dengan cara mewarisi *identifier* kedua entitas awal dan membuat keduanya *identifier* unik entitas penghubung ini. Sering kali akan ada atribut lain yang dimiliki oleh entitas penghubung tersebut. Entitas Kelas dalam contoh model data kita dapat menjadi contoh entitas penghubung. Apabila tidak ada entitas penghubung yang dapat diciptakan, relasi M-N tetap harus diubah untuk menghindari kesulitan dalam konversi model data menjadi skema database fisik.

4. Menterjemahkan Model Data

Setelah sebuah model data dinormalisasikan dan siap diubah menjadi database fisik, ada beberapa langkah penterjemahan yang harus dilakukan:

1. Setiap entitas menjadi tabel tersendiri.
2. Setiap atribut menjadi kolom-kolom tabel tersebut, dengan tipe data yang sesuai.
3. *Identifier* entitas tersebut menjadi kolom ID yang tidak boleh kosong (`NOT NULL`) dan berisi indeks yang unik. ID unik ini dalam database dinamakan *primary key*.
4. Relasi diterjemahkan menjadi *foreign key*.

Skema fisik model data yang dihasilkan tampak dalam Gambar 4. Perhatikan penghilangan spasi, penentuan tipe data dan penyeragaman kapitalisasi untuk portabilitas skema untuk digunakan dalam berbagai implementasi RDBMS yang mungkin berbeda dalam *case-sensitivity*.



Gambar 4: Skema fisik final dari contoh model data.

Dan perintah SQL untuk menciptakan ketiga tabel tersebut adalah:

```
CREATE TABLE dosen (
  nip INTEGER NOT NULL AUTO_INCREMENT,
  nama_lengkap VARCHAR(20) NOT NULL,
  nomor_kontak VARCHAR(20) NULL,
  PRIMARY KEY(nip)
)
TYPE=InnoDB;

CREATE TABLE kelas (
  kode INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  nama_matakuliah INTEGER UNSIGNED NOT NULL,
```

```
  id_tugas_matakuliah INTEGER NOT NULL,
  id_dosen INTEGER NOT NULL,
  ruang_kuliah VARCHAR(5) NULL,
  PRIMARY KEY(kode, nama_matakuliah, id_tugas_matakuliah),
  INDEX kelas_FKIndex1(id_dosen),
  INDEX kelas_FKIndex2(id_tugas_matakuliah)
)
TYPE=InnoDB;

CREATE TABLE tugas_matakuliah (
  id INTEGER NOT NULL,
  deskripsi TEXT NULL,
  batas_penyerahan DATETIME NULL,
  PRIMARY KEY(id)
)
TYPE=InnoDB;
```

Script SQL di atas menggunakan tipe data dan konfigurasi tabel yang didukung oleh MySQL. Deklarasi `TYPE=InnoDB` untuk setiap tabel adalah agar MySQL menggunakan InnoDB yang mendukung penggunaan *foreign key*. Tanpa deklarasi tersebut MySQL secara default akan menggunakan mesin penyimpan MyISAM yang tidak dapat mendukung *foreign key*.

4.1. Foreign Key

Beberapa catatan khusus mengenai penterjemahan relasi menjadi *foreign key*:

1. Relasi 1-1 diterjemahkan menjadi "*identifier* salah satu tabel menjadi *foreign key* dalam tabel lain". Keputusan mengenai tabel mana yang harus menerima *identifier* tabel lain dapat diambil sesuai keinginan, dan secara teori tidak begitu berpengaruh. Namun, seringkali pertimbangan praktis yang akan menentukan tabel mana yang akan berisi *foreign key*.

2. Khusus untuk penggunaan MySQL sebagai penyimpan database: sampai MySQL versi 5.0 hanya *storage engine* InnoDB yang mendukung penggunaan *foreign key*. Mesin penyimpanan lain yang digunakan MySQL versi 5.0 atau dibawahnya (seperti MyISAM atau BDB) tidak mendukung konfigurasi `FOREIGN KEY` dalam perintah SQL `CREATE TABLE`, dan akan mengabaikannya apabila ia ditemui.

5. Contoh Penggunaan Database Ternormalisasi

Untuk model data non-trivial, database ternormalisasi hampir selalu berisi lebih dari satu tabel, sehingga demi kemudahan pengelolaan, biasanya satu database hanya berisi tabel-tabel yang terkait dalam satu model data saja. Di bawah ini terdapat contoh *query* SQL untuk database ternormalisasi untuk membedakan dengan model data yang hanya menggunakan satu tabel.

```
INSERT INTO dosen(nama_lengkap,nip) VALUES('Jusuf Kalla',127001);
INSERT INTO kelas(id_dosen, kode) VALUES(127001, 1);
INSERT INTO kelas(id_dosen, kode) VALUES(127001, 2);
```

Perintah `INSERT` di atas akan menambah data dosen baru dan dua kelas yang diampu beliau.

```
INSERT INTO tugas_matakuliah(id, deskripsi,batas_penyerahan)
      VALUES(102,
            'Implementasikan sebuah compiler untuk bahasa Small-C,
                  lengkap dengan detail grammar yang digunakan.
                  Compiler tersebut harus menghasilkan kode assembler
8086
                  yang dapat dikompilasi oleh Turbo Assembler atau
NASM.',
            '2006-12-01');
UPDATE kelas,dosen SET id_tugas_matakuliah=102
      WHERE kelas.id_dosen=dosen.nip AND
            dosen.nama_lengkap='Jusuf Kalla';
```

Perintah `INSERT` dan `UPDATE` di atas menambah data tugas baru dari dosen tertentu dan memperbarui data untuk setiap kelas yang diampu dosen tersebut.

```
SELECT t.deskripsi, t.batas_penyerahan
      FROM tugas_matakuliah t, kelas k, dosen d
      WHERE k.id_tugas_matakuliah=t.id AND
            k.id_dosen=d.nip AND
            d.nama_lengkap='Jusuf Kalla';
```

Perintah `SELECT` di atas akan menampilkan informasi tentang deskripsi sebuah tugas yang diberikan pada kelas-kelas matakuliah yang diampu oleh dosen tersebut, ditambah dengan informasi tanggal penyerahan tugas terakhir.

6. Penutup

Model data di atas dan semua *query* yang diberikan kepada database yang dihasilkan masih sangat sederhana (Salah satunya adalah penggunaan asumsi bahwa beberapa kelas untuk matakuliah tertentu hanya diampu oleh satu dosen dan mendapat tugas satu-persatu secara sekuensial – asumsi yang terlalu baik hati) dan tidak sepenuhnya memanfaatkan semua kemudahan dan potensi yang ditawarkan baik oleh SQL maupun oleh fasilitas-fasilitas setiap implementasi RDBMS yang mendukung SQL.

Demi keringkasan dan kesederhanaan contoh, saya tidak membahas antara lain mengenai relasi turunan (*inheritance*) serta berbagai notasi grafis model data lain seperti ER dan berbagai turunannya.

Aplikasi yang saya gunakan untuk menggambar model data dan menterjemahkannya menjadi skema database fisik adalah DBDesigner 4 dari fabForce yang secara eksplisit mendukung MySQL dan berbagai mesin penyimpanan yang digunakan produk tersebut. *Script* SQL `CREATE TABLE` yang dihasilkan kemudian ditulis ke *clipboard* untuk di-*paste* ke artikel ini dan ke file .sql untuk dieksekusi oleh MySQL (versi 5.0). Batasan-batasan terkait mesin penyimpanan yang saya paparkan di atas tidak berlaku pada implementasi RDBMS lain (tabel dalam PostgreSQL dan SQL Server, misalnya, secara *default* mendukung *foreign key*).

Sebagai latihan, Anda dapat mencoba memperluas model data di atas atau membuat model data sendiri, lalu menggunakan perintah *query* yang lebih efisien; salah satu contoh adalah perintah `SELECT` dengan modifier `INNER` / `OUTER JOIN` yang memberi Anda kendali yang lebih besar dan fleksibel untuk mendapat set hasil yang Anda inginkan dari sumber data multi-tabel.