Lantip Diat Prasojo

PERANCANGAN DATABASE SISTEM INFORMASI MANAJEMEN PENDIDIKAN

dengan DBMS Microsoft (Acces dan SQL Server)

UNY PRESS

PERANCANGAN DATABASE SISTEM INFORMASI MANAJEMEN PENDIDIKAN DENGAN DBMS MICROSOFT (ACCES DAN SQL SERVER)

LANTIP DIAT PRASOJO

Undang-undang Republik Indonesia Nomor 19 Tahun 2002 tentang Hak Cipta

Lingkup Hak Cipta

Pasal 2:

1. Hak Cipta merupakan hak eksklusif bagi Pencipta atau Pemegang Hak Cipta untuk mengumumkan atau memperbanyak ciptaannya, yang timbul secara otomatis setelah suatu ciptaan dilahirkan tanpa mengurangi pembatasan menurut peraturan perundang-undangan yang berlaku.

Ketentuan Pidana

Pasal 72:

1. Barangsiapa dengan sengaja atau tanpa hak melakukan perbuatan sebagaimana dimaksudkan dalam Pasal 2 ayat (1) atau pasal 49 ayat (1) dan (2) dipidanakan dengan pidana penjara masing-masing paling singkat 1 (satu) bulan dan/atau denda paling sedikit Rp1.000.000,00 (satu juta rupiah), atau pidana penjara paling lama 7 (tujuh) tahun dan/atau denda paling banyak Rp 5.000.000.000,00 (lima miliar rupiah).
2. Barangsiapa dengan sengaja menyiarkan, memamerkan, mengedarkan, atau menjual kepada umum suatu ciptaan atau barang hasil Pelanggaran Hak Cipta atau Hak Terkait sebagaimana dimaksudkan dalam ayat (1) dipidanakan dengan pidana penjara paling lama 5 (lima) tahun dan/atau denda paling banyak Rp 500.000.000,00 (lima ratus juta rupiah).

PERANCANGAN DATABASE SISTEM INFORMASI MANAJEMEN PENDIDIKAN DENGAN DBMS MICROSOFT (ACCES DAN SQL SERVER)

LANTIP DIAT PRASOJO

2014

PERANCANGAN DATABASE
SISTEM INFORMASI
MANAJEMEN PENDIDIKAN
DENGAN DBMS MICROSOFT
(ACCES DAN SQL SERVER)

Oleh:
LANTIP DIAT PRASOJO
ISBN: 978-602-7981-35-5

Edisi Pertama

Diterbitkan dan dicetak oleh:
UNY Press
Jl. Gejayan, Gg. Alamanda, Komplek Fakultas Teknik UNY
Kampus UNY Karangmalang Yogyakarta 55281
Telp: 0274 – 589346
Mail: unypress.yogyakarta@gmail.com

Penyunting Bahasa: Alan Wijanarko
Desain Sampul: M. Yazid
Tata Letak: Yudi Rahman

Isi di luar tanggung jawab percetakan

Lantip Diat Prasojo
PERANCANGAN DATABASE SISTEM
INFORMASI MANAJEMEN PENDIDIKAN
DENGAN DBMS MICROSOFT (ACCES DAN SQL SERVER)
-Ed.1, Cet.1.- Yogyakarta: UNY Press 2014
xii+ 152 hlm; 16 x 23 cm
ISBN: 978-602-7981-35-5
1. perancangan database sistem informasi manajemen pendidikan dengan dbms microsoft (acces dan sql server)

KATA PENGANTAR

*Alhamdulillahi robbil 'aalamiin...*penulis panjatkan ke hadirat Allah SWT., atas segala nikmat yang diberikan sehingga buku ini dapat terselesaikan tanpa rintangan yang berarti.

Belakangan ini sudah banyak buku yang membahas tentang Sistem Informasi Manajemen (SIM), tetapi masih ditemukan kekurangan di berbagai sisi. Hampir seluruh buku yang beredar hanya membahas salah satu dari bagian SIM, sehingga untuk menguasainya, pembaca harus membeli banyak buku, belum lagi dihadapkan dengan masalah kualitas isi buku, tentunya hal ini merupakan investasi yang sangat mahal.

Buku ini hadir untuk memberi pemahaman tentang Perancangan Sistem Informasi Manajemen berbasis Microsoft Acces dan SQL Server, sebelum lebih jauh mendalaminya. Pembahasan yang singkat, padat, jelas, dan disertai berbagai ilustrasi gambar, serat contoh-contoh aplikasi SIM.

Buku ini ditujukan untuk berbagai kalangan pembaca, mulai dari siswa SMA/SMK, Mahasiswa, Guru, Dosen, maupun kalangan umum yang tertarik dengan teknologi informasi. Buku ini dapat dijadikan referensi untuk mata kuliah Sistem Informasi Manajemen, Manajemen Informasi, Teknologi Informasi dan Komunikasi (TIK), dan berbagai mata kuliah sejenis lainnya.

Secara singkat, buku ini membahas berbagai hal yang terkait dengan Perancangan database SIM, seperti Sistem Database Relasional, Membuat Database SIM Perpustakaan dengan Microsoft Access, Membuat Form dan Laporan dengan Microsoft Access, Instalasi Microsoft SQL Server dan Konfigurasinya, Administrasi Database Microsoft SQL Server.

Pada kesempatan ini penulis ingin menghaturkan rasa terima atas kontribusi berbagai pihak, diantaranya adalah:

- Jajaran Pimpinan fakultas Ilmu Pendidikan dan UNY.
- Prodi Manajemen Pendidikan Fakultas Ilmu Pendidikan UNY.

- Pihak Penerbit UNY Press.
- Orang tua dan Keluargaku tercinta

Anda dapat berinteraksi dengan memberikan pertanyaan atau saran mengenai materi buku demi perbaikan isi buku pada edisi berikutnya melalui alamat e-mail: **lantip1975@gmail.com**

Yogyakarta, Oktober 2014

Lantip Diat Prasojo

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

1 Sistem Database Relasional

Hal mendasar dari struktur *database* adalah model data, yaitu sekumpulan cara untuk mendeskripsikan berbagai data, hubungannya satu sama lain, semantiknya, serta batasan konsistensi. Untuk memperlihatkan konsep dari model data, berikut ini akan diuraikan dua model data yang paling populer, yaitu **Model ERD** (*Entity Relationship Diagram*) dan **Model Relasional**. Keduanya menyediakan cara untuk mendeskripsikan perancangan *database* pada peringkat logika. Selain model data relasional dan ERD, sebenarnya ada banyak model data lain, seperti Hierarkhis, Jaringan, dan Berorientasi Objek, dengan berbagai keunggulan dan kekurangan masing-masing.

1.1 Model ERD

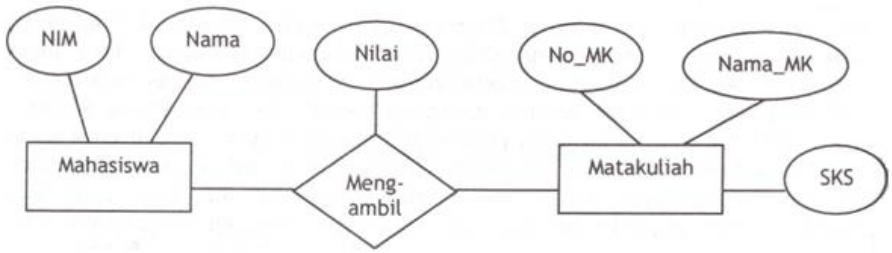
Model ERD (*entity relationship diagram*) dibuat berdasarkan anggapan bahwa dunia nyata terdiri dari koleksi objek-objek dasar yang dinamakan entitas (*entity*) serta hubungan (*relationship*) antara entitas-entitas itu. Entitas adalah "sesuatu" atau "objek" pada dunia nyata yang dapat dibedakan antara satu dengan yang lainnya, yang bermanfaat bagi aplikasi yang akan kembangkan. Sebagai contoh, setiap **orang** adalah entitas dan **rekening bank** dapat dipertimbangkan sebagai sebuah entitas.

Entitas dalam *database* dideskripsikan berdasarkan atributnya. Sebagai contoh, **nomor rekening** membedakan suatu rekening adalah milik seseorang yang menyimpan uangnya dengan rekening milik orang lain di suatu bank tertentu dan nomor-nomor rekening tersebut merupakan atribut dari entitas rekening yang bersangkutan. Dalam hal ini, nomor rekening secara unik membedakan sebuah rekening dengan rekening yang lainnya. Beberapa rekening mungkin memiliki saldo yang sama, tetapi mereka pasti memiliki nomor rekening yang berbeda.

Relationship adalah hubungan antara beberapa entitas. Sebagai contoh, **mahasiswa** "memiliki" **arangtua**, "memiliki" menjelaskan hubungan tertentu antara mahasiswa dengan orangtuanya. Dalam hal ini, himpunan semua entitas dengan tipe yang sama dan himpunan

semua hubungan antar entitas dirujuk sebagai himpunan entitas (*entity set*) dan himpunan relasi (*relationship set*). Secara skematik, *database* dapat dideskripsikan secara grafis dengan ERD yang memiliki komponen-komponen utama sebagai berikut:

- **Empat-persegi-panjang**, yang menggambarkan himpunan entitas.
- **Elips**, yang menggambarkan atribut.
- **Jajaran genjang**, yang menggambarkan relasi/hubungan antarentitas.
- **Garis**, yang menyatukan atribut-atribut pada entitas tertentu serta menyatukan entitas-entitas dalam suatu relasi tertentu.



Gambar 1.1: Penggambaran Diagram ER

Sebagai contoh, Gambar 1.1 menunjukkan Diagram ER yang menggambarkan hubungan antara entitas **Mahasiswa** dengan **Matakuliah**. Model pada Gambar 1.1 di atas adalah model penggambaran Diagram ER secara umum. Penggambaran itu belum memperlihatkan kardinalitas, yaitu jumlah suatu entitas yang berhubungan dengan entitas yang lainnya.

1.2 Model Data Relasional

Model *database* relasional menggunakan sekumpulan tabel berdimensi dua (biasa disebut tabel) yang merupakan tempat data disimpan. Masing-masing tabel dalam model *database* relasional terdiri dari baris dan kolom. Selain itu pada model *database* relasional juga terdapat istilah *key* (kunci). *Key* adalah satu atau gabungan beberapa atribut yang dapat membedakan semua baris data dalam tabel secara unik. Artinya jika suatu atribut dijadikan *key,* maka tidak boleh ada dua

atau lebih baris dengan nilai yang sama untuk kolom tersebut. Ada tiga macam *key* yang dapat diterapkan pada suatu tabel, yaitu:

- **Superkey**, yaitu merupakan satu atau lebih atribut yang dapat membedakan setiap baris data dalam sebuah tabel secara unik.
- **Candidat-Key**, yaitu merupakan kumpulan atribut minimal yang dapat membedakan setiap baris data dalam sebuah tabel secara unik. Sebuah *candidat-key* tidak boleh berisi atribut atau kumpulan atribut yang telah menjadi *super-key*.
- **Primary-Key**, yaitu merupakan *candidat-key* yang unik yang digunakan sebagai acuan dan kunci utama. Perhatikan struktur dan contoh data pada Tabel **Mahasiswa**, **Matakuliah**, dan **Kuliah** (representasi dari hubungan **mengambil** antara entitas **Mahasiswa** dan **Matakuliah** pada Diagram ER – Gambar 1.1).

Tabel 1.1: Tabel **Mahasiswa**

NIM	Nama
123456	Riyanto
123457	Sholihun
123458	Sugiharti

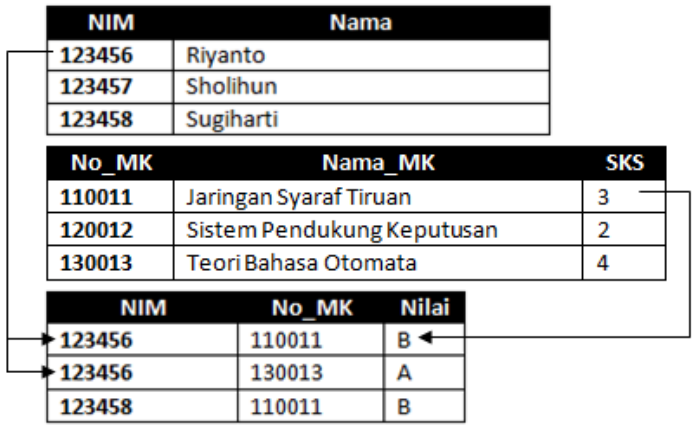
Tabel 1.2: Tabel **Matakuliah**

No_MK	Nama_MK	SKS
110011	Jaringan Syaraf Tiruan	3
120012	Sistem Pendukung Keputusan	2
130013	Teori Bahasa Otomata	4

Tabel 1.3: Tabel **Kuliah**

NIM	No_MK	Nilai
123456	110011	B
123456	130013	A
123458	110011	B

Ketiga tabel di atas memperlihatkan **seorang** mahasiswa yang mengambil **beberapa** matakuliah, misalnya **Riyanto** (NIM=123456) mengambil matakuliah **Jaringan Syaraf Tiruan** (No_MK=**110011**) dan mendapatkan indeks nilai **B**, dan juga mengambil matakuliah **Sistem Pendukung Keputusan** (No_MK=**120012**) dan mendapatkan nilai **A**.



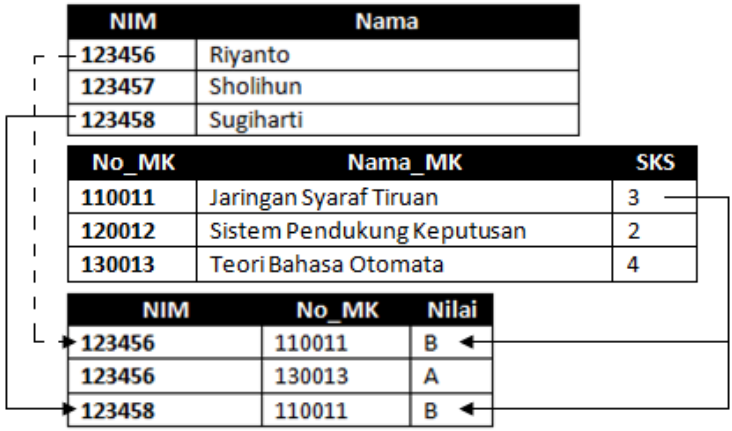
NIM	Nama
123456	Riyanto
123457	Sholihun
123458	Sugiharti

No_MK	Nama_MK	SKS
110011	Jaringan Syaraf Tiruan	3
120012	Sistem Pendukung Keputusan	2
130013	Teori Bahasa Otomata	4

NIM	No_MK	Nilai
123456	110011	B
123456	130013	A
123458	110011	B

Gambar 1.2: Satu mahasiswa mengambil beberapa matakuliah

Di sisi yang lain, ada **beberapa** mahasiswa yang mengambil **satu** matakuliah yang sama, misalnya **Riyanto** (NIM=123456) dan **Sugiharti** (NIM=**123458**) mengambil matakuliah yang sama, yaitu **Jaringan Syaraf Tiruan** (No_MK=**110011**) dan keduanya mendapatkan indeks nilai **B**.



NIM	Nama
123456	Riyanto
123457	Sholihun
123458	Sugiharti

No_MK	Nama_MK	SKS
110011	Jaringan Syaraf Tiruan	3
120012	Sistem Pendukung Keputusan	2
130013	Teori Bahasa Otomata	4

NIM	No_MK	Nilai
123456	110011	B
123456	130013	A
123458	110011	B

Gambar 1.3: Beberapa mahasiswa mengambil matakuliah yang sama

Model relasional adalah contoh model berbasis *record*. Dinamakan seperti itu karena *database* memiliki struktur *record* berformat tertentu dimana masing-masing isinya memiliki tipe-tipe yang berbeda, misalnya tipe data untuk **NIM** adalah **string[8]** tentu berbeda dengan tipe data untuk **Nama** yang mungkin juga bertipe data **string** yang panjangnya tidak ditentukan, bergantung pada komputer tempat aplikasi diimplementasikan). Dalam hal ini, setiap kolom pada tabel-tabel mencerminkan atribut-atribut entitas yang bersangkutan yang sering di jumpai di model konseptual ERD.

Dapat dilihat bahwa tabel-tabel dapat disimpan dalam file-file. Sebagai contoh, karakter-karakter khusus, misalnya tanda koma "," mungkin dapat digunakan untuk memisahkan atribut-atribut yang berbeda dalam suatu *record*, dan karakter-karakter khusus yang lainnya dapat digunakan untuk memisahkan suatu *record* dengan *record* yang lainnya. Model relasional menyembunyikan implementasi aras rendah *database* dari pengembang dan pengguna aplikasi basis.

Model relasional adalah abstraksi pada peringkat yang lebih rendah dari ERD. Perancang *database* umumnya pertama kali menggunakan ERD kemudian menerjemahkannya ke model relasional untuk kemudian diimplementasikan di sistem *database* yang digunakan.

1.3 Keunggulan *Database* Relasional

Pendekatan *database* menawarkan keunggulan-keunggulan dibandingkan sistem pemrosesan berkas tradisional. Keunggulan-keunggulan yang dimaksud adalah sebagai berikut:

1) Kemandirian Program dan Data. Pemisahan deskripsi data (*meta data*) dari program aplikasi yang menggunakan data tersebut dinamakan kemandirian data (*data independence*). Dengan pendekatan *database*, deskripsi data disimpan di lokasi terpusat yang dinamakan *repository*. Karakteristik sistem *database* ini mengijinkan organisasi data berubah (sampai batas-batas tertentu) tanpa mempengaruhi program aplikasi yang memproses data tersebut (tidak memerlukan pemrograman ulang – *reprogramming*) .
2) Mengurangi Pengulangan Data (Redundansi) yang Tidak Periu. Sasaran perancangan dengan pendekatan *database* adalah menyatukan berkas-berkas data pada suatu struktur logika yang

tunggal. Setiap fakta primer direkam pada hanya satu tempat di *database*. Misalnya, dari contoh di atas, IP seorang mahasiswa hanya tersimpan pada *Tabel Mahasiswa*. Pendekatan *database* tentu saja tidak menghilangkan redundansi data sama sekali, tetapi ia mengijinkan perancang secara hati-hati mengendalikan redundansi. Sebagai contoh (Lihat Gambar 1.4) setiap NIM pada *Tabel Pengambilan Matakuliah* harus berpasangan dengan NIM pada *Tabel Mahasiswa*. Hal ini menegaskan bahwa ada relasi antara *Tabel Mahasiswa* dan *Tabel Pengambilan Matakuliah*.

3) Memperbaiki Konsistensi Data. Dengan mengendalikan redundansi, secara dramatis mengurangi kesempatan untuk terjadinya ketidakkonsistenan data. Misalnya, pada *Tabel Mahasiswa*, setiap NIM berpasangan dengan nama mahasiswa tertentu. Jika ingin merubah nama mahasiswa tertentu, maka cukup melakukannya hanya pada *Tabel Mahasiswa*. Pada tabel **Kuliah**, tidak ditemukan kolom nama sehingga – seperti diungkapkan sebelumnya – tidak perlu merubah nama mahasiswa pada tabel tersebut karena hal itu telah terwakili dengan perubahan nama pada tabel **Mahasiswa**.
4) Memperbaiki Kesempatan Berbagi Data (*Data Sharing*). *Database* dirancang untuk berbagi sumberdaya data dalam organisasi. Pengguna dengan hak tertentu (yang diatur sebelumnya oleh administrator *database*) dapat mengakses bagian tertentu dalam *database*, di manapun pengguna tersebut berada dalam organisasi.
5) Menambah Produktivitas Pengembangan Program Aplikasi. Salah satu keunggulan pendekatan *database* adalah pengurangan waktu dan biaya untuk mengembangkan aplikasi bisnis yang baru. Ada 2 alasan yang memungkinkan aplikasi *database* dikembangkan dengan cara yang lebih cepat dibandingkan aplikasi yang mengakses berkas data pada pendekatan tradisional, yaitu:
    - Dengan mengasumsikan bahwa *database* telah dirancang dan diimplementasikan, pemrogram cukup berkonsentrasi pada fungsi spesifik yang dibutuhkan oleh aplikasi baru tanpa perlu memusingkan perancangan berkas data atau rincian implementasi aras rendah.

- Sistem manajemen *database* menyediakan sejumlah *tool* seperti pembuatan form dan generator laporan (*report generator*) dan bahasa tingkat tinggi dapat mengotomatisasi beberapa aktivitas perancangan *database* dan implementasinya.

6) Memaksakan Standar. Saat pendekatan *database* diimplementasikan dengan dukungan manajemen secara penuh, administrasi data dan administrasi *database* dapat dilakukan oleh seseorang yang memang ditunjuk untuk hal itu. Dengan pengelolaan *database* pada satu tangan, ia dapat diberi tanggung jawab untuk memaksakan standar *database* untuk keseluruhan organisasi. Standar yang dimaksud antara lain adalah: tata cara penamaan, standar kualitas data, serta prosedur yang seragam untuk mengakses, memperbaharui, serta melindungi data. *Repository* memungkinkan administrator *database* memaksakan standar itu dengan sekumpulan kakas yang berdaya guna.
7) Memperbaiki Kualitas Data. Kualitas data yang rendah selama ini telah menjadi pusat perhatian dalam organisasi-organisasi skala besar. Pendekatan *database* menyediakan sejumlah kakas dan proses untuk memperbaiki kualitas data. Dua diantaranya adalah:
   - Perancang *database* dapat menspesifikasi batasan-batasan integritas (untuk memelihara konsistensi data) yang dapat dipaksakan oleh sistem *database*.
   - Salah satu sasaran dari data *warehousing* adalah meletakkan hanya data-data yang berkualitas pada data *warehouse*.
8) Memperbaiki Akses Data. Dengan *database* bertipe relasional, pengguna tanpa pengetahuan dan pengalaman pemrograman dapat memanggil dan menampilkan data-data tertentu sesuai hak yang dimilikinya, meskipun cara mendapatkannya mungkin melewati batas-batas departemen dimana pengguna itu berada. Suatu cara untuk melakukannya adalah hanya dengan menuliskan pernyataan-pernyataan SOL yang implementasinya dilakukan oleh sistem *database*, tanpa pengguna perlu tahu teknik-teknik algoritma-algoritma yang diadaptasi dan

diimplementasi sistem *database* untuk mendapatkan data-data itu.

9) Mengurangi Biaya Pemeliharaan Program. Data yang tersimpan harus diubah karena beberapa alasan: tipe data baru ditambahkan, format data berubah, dan sebagainya. Pada sistem pemrosesan berkas, deskripsi data dan logika untuk mengakses data bersatu dalam suatu program aplikasi. Akibatnya, perubahan format data dan metoda akses data akan mengakibatkan pemrogram harus merubah program.

1.4 Bahasa *Database*

Sistem *database* menyediakan bahasa untuk mendefinisikan *database* (*data definition language* – DDL) dan memanipulasi *database* (*data manipulation language* – DML) untuk melakukan operasi-operasi tertentu pada *database*. Dalam prakteknya, kedua jenis bahasa *database* itu tidak benar-benar dapat dipisahkan secara tegas. Saat ini keduanya merupakan bagian dari bahasa *database* tunggal yang disebut SQL (*structured query language*) yang merupakan bahasa *database* standar untuk *database* bertipe relasional.

1.4.1 DDL (*Data Definition Language*)

Pada umumnya mendefinisikan skema *database* dengan sekumpulan definisi yang diekspresikan dengan bahasa khusus yang dinamakan DDL (*Data Definition Language*). Sebagai contoh, tabel **Mahasiswa** mungkin didefinisikan dengan cara berikut:

```
create table Mahasiswa
(NIM char(8), Nama char(30), IP real)
```

Eksekusi pernyataan DDL di atas menciptakan tabel **Mahasiswa**. Sebagai tambahan, pernyataan di atas juga menciptakan apa yang dinamakan kamus data (*data dictionary*). Kamus data adalah suatu himpunan dari *metadata* (suatu data yang menerangkan data lainnya). Skema tabel adalah suatu contoh dari *metadata*. Sistem *database* akan membaca kamus data sebelum membaca atau memodifikasi data yang sebenarnya.

Menentukan spesifikasi struktur tempat penyimpanan dan metode akses menggunakan pernyataan khusus DDL dinamakan data ***Storage and Definition Language***. Pernyataan-pernyataan ini mendefinisikan rincian implementasi skema *database*, yang pada umumnya tersembunyi dari pengguna.

Data yang tersimpan pada *database* biasanya memiliki batasan-batasan (*constraint*) tertentu (Misalnya nilai IP tidak boleh lebih kecil dari 0 dan tidak boleh lebih besar dari 4). Dalam hal ini DDL dapat digunakan untuk menentukan batasan-batasan *database* tersebut, sistem *database* memeriksa batasan-batasan saat *database* disisipkan dan diperbaharui.

1.4.2 DML (*Data Manipulation Language*)

DML (*Data Manipulation Language*) adalah bahasa yang memungkinkan pengguna untuk mengakses atau memanipulasi data dalam sistem *database* yang bertipe relasional. Pada dasarnya ada 2 jenis DML, yaitu:

- DML Prosedural yang menghendaki pengguna untuk menspesifikasi data apa yang diperlukan dan bagaimana cara mendapatkan data itu. Ini dapat dilakukan dengan bahasa-bahasa pemrograman yang mampu mengakses *database* (Misalnya: PHP dan ASP).
- DML Deklaratif (DML Non Prosedural) yang menghendaki pengguna untuk menspesifikasi data apa yang diperlukan tanpa harus menspesifikasi bagaimana caranya mendapatkannya. Contoh dari DML Non Prosedural ini adalah SOL (*Structured Query Language*).

DML Deklaratif pada umumnya relatif mudah dipelajari dan digunakan dibandingkan DML Prosedural karena tidak harus menentukan bagaimana caranya mendapatkan data yang dibutuhkan, sistem *database* relasional akan menentukan sendiri cara-cara yang efisien untuk mendapatkan data tersebut, menyangkut algoritma yang akan digunakan, strategi-strategi untuk mengoptimalkan kinerja proses, dan sebagainya. Manipulasi data pada *database* umumnya meliputi hal-hal sebagai berikut ini.

a. Pemanggilan informasi yang tersimpan pada *database* (*query*).
b. Penambahan informasi baru pada *database*.

c. Penghapusan informasi yang tidak diperlukan lagi pada *database*.
d. Modifikasi informasi yang ada pada *database*.

1.5 Perintah SQL

SQL (dibaca "ess-que-el") digunakan untuk berkomunikasi dengan *database*. Menurut ANSI (American National Standards Institute), SQL merupakan bahasa standar untuk sistem manajemen *database* relasional. Perintah SQL digunakan untuk melakukan tugas-tugas seperti perbaruan data, atau mengambil data dari *database*. Beberapa sistem manajemen *database* relasional umum yang menggunakan SQL adalah: **Oracle**, **Sybase**, **Microsoft SQL Server**, **Access**, **MySQL** dan lain-lain. Meskipun sebagian besar sistem *database* menggunakan SQL, kebanyakan dari mereka juga memiliki ekstensi tambahan milik mereka sendiri yang biasanya hanya digunakan pada sistem mereka. Meskipun demikian, perintah-perintah SQL standar seperti **Select**, **Insert**, **Update**, **Delete**, **Create** dan **Drop**, dapat digunakan pada seluruh sistem *database* relasional yang ada saat ini. Pada bab ini akan dibahas dasar-dasar perintah SQL serta penggunaanya dalam operasi *database*.

Lebih lanjut, *query* adalah pernyataan yang meminta pemanggilan informasi tertentu dari *database*. Sebagian dari DML dinamakan bahasa *query*. Meskipun tidak terlalu tepat, orang sering menyebut seluruh DML sebagai bahasa *query*. Suatu pernyataan *query* untuk menampilkan data-data mahasiswa yang memiliki IP (indeks prestasi) lebih besar atau sama dengan 2.75 adalah sebagai berikut:

```
SELECT * FROM Mahasiswa WHERE IP >= 2.75
```

Perhatikan! Pada pernyataan SOL di atas hanya menyebutkan informasi-informasi yang dibutuhkan dan tidak menyebutkan bagaimana caranya mendapatkannya. Pada dasarnya pernyataan SOL di atas oleh kompiler akan diterjemahkan ke algoritma-algoritma tertentu yang memungkinkan akses paling efisien pada *database*. Berikut disajikan beberapa perintah SQL yang akan dibahas pada bab ini.

1.5.1 Perintah SQL Dasar

- **SELECT**, digunakan untuk memfilter atribut-atribut dari relasi (tabel) berdasarkan kondisi yang mengikutinya.
- **FROM**, digunakan untuk menunjukkan dari relasi mana data yang akan difilter.
- **WHERE**, digunakan untuk membuat suatu kondisi.
- **GROUP BY**, digunakan untuk mengelompokkan data berdasarkan atribut tertentu.
- **HAVING**, digunakan untuk mendukung klausa **GROUP BY**, yakni untuk menentukan kondisi bagi klausa **GROUP BY**.
- **AVG**, digunakan untuk menghitung rata-rata.
- **COUNT**, digunakan untuk menghitung cacah data.
- **MAX**, digunakan untuk memperoleh nilai terbesar
- **MIN**, digunakan untuk memperoleh nilai terkecil.
- **SUM**, digunakan untuk memperoleh jumlahan data.
- Dan berbagai perintah SQL lainnya.

1.5.2 Sub-*Query*

Subquery berarti *query* di dalam *query*. Dengan menggunakan *subquery*, hasil *query* akan menjadi bagian dari *query* lain. *Subquery* terletak di dalam klausa WHERE atau HAVING. Pada klausa WHERE, *subquery* digunakan untuk memilih baris-baris tertentu, yang kemudian digunakan oleh *query*. Sedangkan pada klausa HAVING, *subquery* digunakan untuk memilih kelompok baris, yang kemudian digunakan oleh *query*.

- **EXISTS**, digunakan untuk memeriksa keadaan baris yang dihasilkan *query* terhadap yang dihasilkan oleh *subquery*.
- **ANY**, digunakan berkaitan dengan *subquery*, hampir mirip dengan memilih tetapi dengan operasi **OR** (lihat contoh penerapan perintah ANY di bagian pembahasan).
- **ALL**, digunakan untuk melakukan pembandingan dengan *subquery*. Kondisi dengan ALL menghasilkan nilai true jika *subquery* tidak menghasilkan apapun atau jika pembandingan menghasilkan true untuk setiap nilai *query* terhadap hasil *subquery*.

1.5.3 Perintah SQL untuk Banyak Tabel

Perintah SQL ini digunakan untuk menggabungkan informasi dari berbagai tabel yang terhubung. Perintah yang dimaksud diantaranya adalah sebagai berikut.

- UNION, merupakan operator yang digunakan untuk menggabungkan hasil *query*.
- JOIN, digunakan untuk menggabungkan dua tabel atau lebih dengan hasil berupa gabungan dari kolom-kolom yang berasal dari tabel-tabel tersebut. Pada JOIN sederhana, tabel-tabel digabungkan dan didasarkan pada pencocokan antara kolom pada tabel yang berbeda. Ada beberapa perintah JOIN pada Access, yakni **INNER JOIN**, **LEFT JOIN**, dan **RIGHT JOIN**.

2 Membuat Database SIM Perpustakaan dengan Microsoft Access

Pada bab ini akan dibahas langkah perancangan *database* untuk aplikasi Sistem Informasi Manajemen Perpustakaan (SIM Perpus) dengan **Microsoft Access** (selanjutnya disebut "**Access**").

2.1 Daftar Informasi Yang Dibutuhkan

Langkah awal dalam perancangan suatu *database* adalah mengumpulkan terlebih dahulu segala informasi yang berkenaan dengan sistem *database* yang akan dirancang. Berdasarkan informasi-informasi yang ada inilah selanjutnya dilakukan beberapa normalisasi sehingga menjadi tabel awal rancangan. Dari tabel awal yang di dapat, dilakukan analisis ulang, apakah sudah normal? Jika belum lakukan tahap normalisasi berikutnya sampai terbentuk rancangan daftar tabel yang benar-benar sudah normal, setelah itu baru diimplementasikan ke dalam aplikasi penyedia layanan *database*, seperti Microsoft Access, Interbase, Microsoft SQL Server dan lain-lain. Tetapi dalam hal ini kita menggunakan Microsoft Access. Berikut adalah beberapa informasi minimal yang dapat dikumpulkan berkenaan dengan SIM Perpustakaan Kampus.

Tabel 2.1: Daftar Informasi Tentang Perpustakaan

Judul Buku	No. Telpon Pengarang	NIM Anggota
Deskripsi Buku	Nama Penerbit	Nama Anggota
Kode Buku	Jenis Buku	No. Anggota
Kode Penerbit	Jumlah Buku	Alamat Anggota
Kode Pengarang	No. DCC	Fakultas
Tahun Terbit	Tgl. Pinjam	Jurusan
ISBN	Tgl. Kembali	Prodi
Kota Terbit	Tgl. Harus Kembali	Angkatan
Nama Pengarang	Denda	JenisKelamin

2.2 Rancangan Tabel SIM Perpustakaan

Berdasarkan informasi yang didapat dan dilakukan proses normalisasi maka dihasilkan daftar rancangan tabel berikut ini.

TB.SIRKULASI

Field	Tipe	Ket
Kode_Buku	Number(10)	+
No_Anggota	Number(10)	+
Tgl_Pinjam	Date	
Tgl_Harus_kembali	Date	
Tgl_Kembali	Date	
Denda	Number(10)	

TB.DAFTAR_BUKU

Field	Tipe	Ket
Kode_buku	Number(10)	*
ISBN	Text(20)	+,U

TB.INFO_BUKU

Field	Tipe	Ket
ISBN	Text(20)	*
DCC	Text(20)	
Judul	Text(100)	
Deskripsi	Text(255)	
Kode_Pengarang	Number(10)	+
Kode_Penerbit	Number(10)	+
Tahun_Terbit	Number(4)	

TB.JENIS_BUKU

Field	Tipe	Ket
Kode_DCC	Text(20)	*
Jenis_Buku	Text(50)	

TB.ANGGOTA

Field	Tipe	Ket
No_Anggota	Number(10)	*
NIM	Text(20)	
Nama	Text(40)	
Alamat	Text(100)	
Jenis_kelamin	Number(1)	0=P; 1=L;
Anggkatan	Number(4)	
Kode_Prodi	Number(10)	+

TB.PRODI

Field	Tipe	Ket
Kode_Prodi	Number(10)	*
Nama_prodi	Text(50)	
Jurusan	Text(50)	
Fakultas	Text(50)	

TB.PENGARANG

Field	Tipe	Ket
Kode	Number(2)	*
Nama	Text(50)	
No_Telpon	Text(30)	

TB.PENERBIT

Field	Tipe	Ket
Kode	Number(10)	*
Nama	Text(50)	
Kota	Text(50)	

Keterangan *:*

* : Primary key.

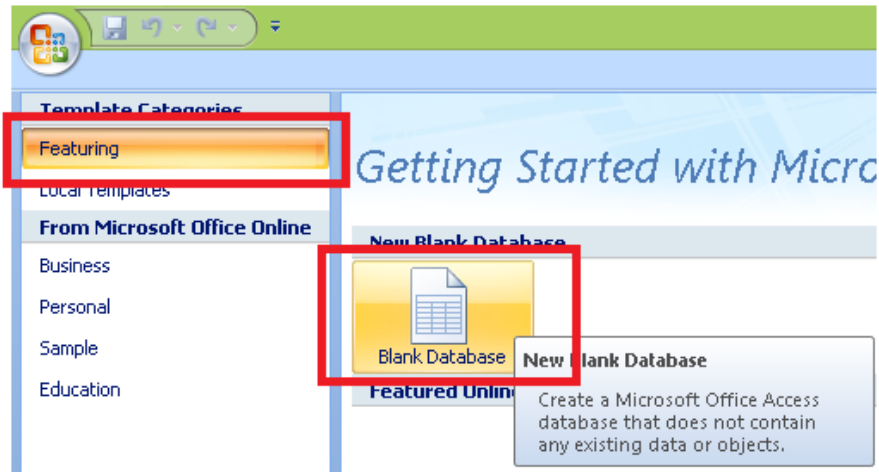
+ : Foreign Key.

U : Unique key.

2.3 Membuat Database Aplikasi SIM Perpustakaan

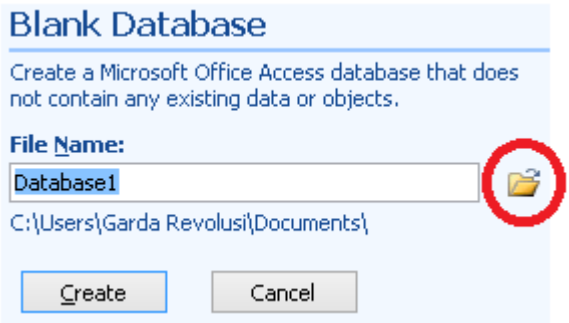
Untuk membuat *database* baru yang masih kosong, langkahnya adalah sebagai berikut:

1) Jalankan program aplikasi Access (dalam hal ini **Microsoft Access 2007**), akses menu **Featuring**.
2) Pada bagian **New Blank Database**, klik ikon **Blank Database** seperti yang ditunjukkan Gambar 2.1.



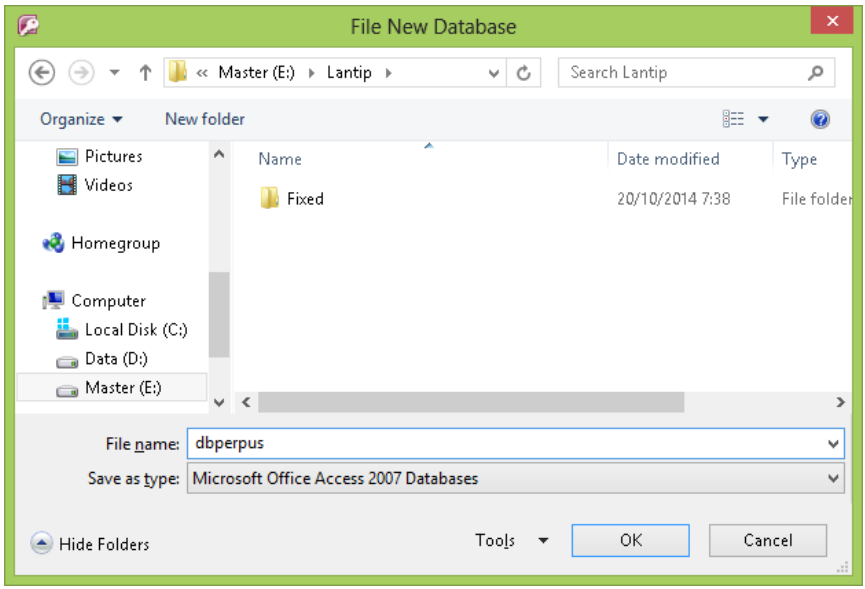
Gambar 2.1: Bagian **New Blank Database**

3) Setelah halaman **Blank Database**, klik tombol untuk menentukan direktori simpan *database* yang akan dibuat.



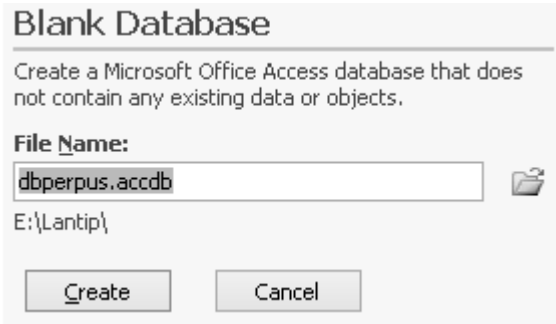
Gambar 2.2: Kotak dialog **Blank Database**

4) Setelah muncul kotak dialog **File New Database** (Gambar 2.3), arahkan ke direktori simpan yang diinginkan (misal "**e:\Lantip**") dan tentukan nama file *database* yang akan dibuat (misal "**dbperpus**"), klik tombol **[OK]**.



Gambar 2.3: Kotak dialog **File New Database**

5) Jika tidak ada kesalahan, maka Anda akan dibawa kembali ke halaman **Blank Database** dengan konfigurasi direktori dan nama yang telah ditentukan. Perhatikan Gambar 2.4 berikut ini.



Gambar 2.4: Halaman **Blank Database** setelah dikonfigurasi

6) Jika berhasil, maka di *header* halaman utama Access akan muncul nama *database* "**dbperpus**" seperti Gambar 2.5 berikut ini.

Gambar 2.5: Header halaman utama Access

2.4 Tabel dan Tipe Data

Setelah selesai membuata database baru yang masih kosong, langkah berikutnya adalah membuat tabel pada databasesesuai rancangan telah ditetapkan. Sebelum lebih teknis membuat tabel, berikut adalah istilah-istilah yang digunakan dalam pembuatan tabel pada Access:

- **Field**, tempat dimana data atau informasi dalam kelompok yang sama atau sejenis dimasukkan. Field pada umumnya tersimpan dalam bentuk kolom secara vertikal pada tabel.
- **Record**, merupakan data lengkap dalam jumlah tunggal, yang biasanya tersimpan dalam bentuk baris secara horisontal pada tabel.

Sedangkan, berikut ini adalah daftar tipe data yang digunakan untuk membuat tabel dalam Access:

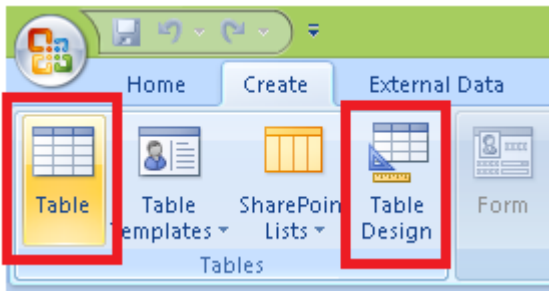
- **Text**, dapat menerima huruf, angka, spasi, dan tanda baca. Sebuah data berisi jenis data teks dapat menampung hingga 255 karakter atau sebanyak lebar yang kita tentukan dalam property FieldSize.
- **Memo**, dapat menerima teks apa saja sebagai catatan atau keterangan dengan panjang maksimal 65535 karakter.
- **Number**, berisi data bilangan yang digunakan untuk perhitungan matematis.
- **Date/Time**, hanya dapat menerima tanggal/waktu. Berisi nilai data tanggal dan waktu untuk tahun 100 sampai dengan 9999.
- **Currency**, berisi nilai uang dan data bilangan yang digunakan dalam perhitungan matematis terhadap data dengan 1 sampai 4 angka di belakang tanda desimal. Tipe data ini memiliki ketelitian sampai 15 digit disebelah kiri dan 4 digit di sebelah kanan tanda desimal.
- **AutoNumber**, berisi bilangan yang berurutan atau bilangan acak yang unik yang secara otomatis diberikan oleh Access jika record baru ditambahkan dalam tabel. Tipe data ini dapat diubah-ubah nilainya oleh *user*. Properti **Fieldsize** dari tipe data ini dapat berupa **Long Integer** atau **Replication ID**.

- **Yes/No**, berisi nilai **YES** atau **NO**, atau *field* yang hanya memiliki dua kemungkinan nilai (**Yes/No**, **True/False**, atau **On/Off**) tergantung kasus yang ditangani. Pada implementasinya, kolom **Jenis Kelamin** juga bisa digunakan tipe ini, misalkan **Yes** untuk "**Laki-laki**" dan **No** untuk "**Perempuan**").
- **OLE Object**, berisi objek yang dikaitkan (linked) atau disisipkan (embedded) ke dalam tabel Access.
- **Hyperlink**, dapat diisi dengan alamat hyperlink (URL) agar bisa terkait dengan object atau data yang tersimpan di lokasi tertentu.
- **Lookup Wizard**, memungkinkan kita untuk memilih nilai dari tabel lain atau dari daftar nilai yang didefinisikan sendiri dengan menggunakan *list box* atau *combo box*.

2.5 Membuat Tabel Baru

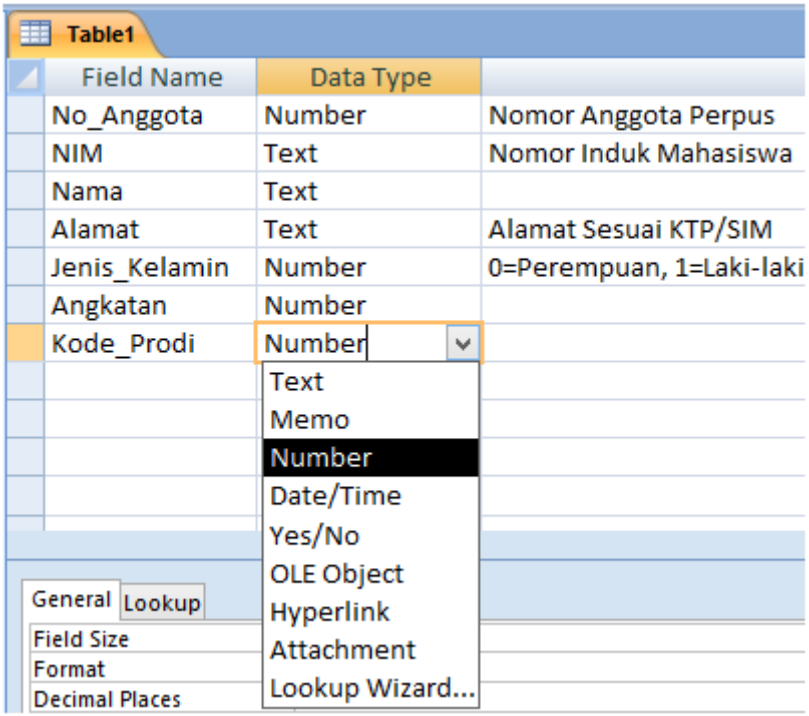
Dalam Access tersedia beberapa fasilitas untuk membuat tabel baru, yaitu **Create Table in Design View**, **Create Table in by Using Wizard**, dan **Create Table in by Entering Data.** Tetapi dalam hal ini akan digunakan fasilitas **Create Table in Design View**. Adapun langkahnya adalah sebagai berikut :

1) Buka file *database* yang telah kita buat (**dbperpus.accdb**), kemudian pada tab **Create**, klik ikon **Table Design**.



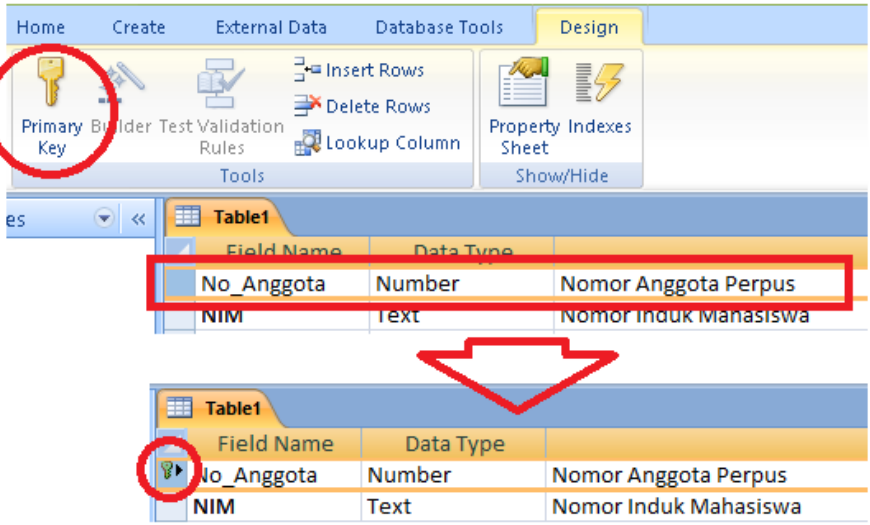
Gambar 2.6: Ikon **Table Design** pada tab **Create**

2) Setelah muncul jendela **Table Design View**, lakukan pendefinisian struktur tabel dengan cara mengisi nama kolom (*field name*) dengan panjang maksimum 64 karakter, tipe data (*data type*) dan keterangan (bila ada).



Gambar 2.7: Proses membuat tabel **tAnggota**

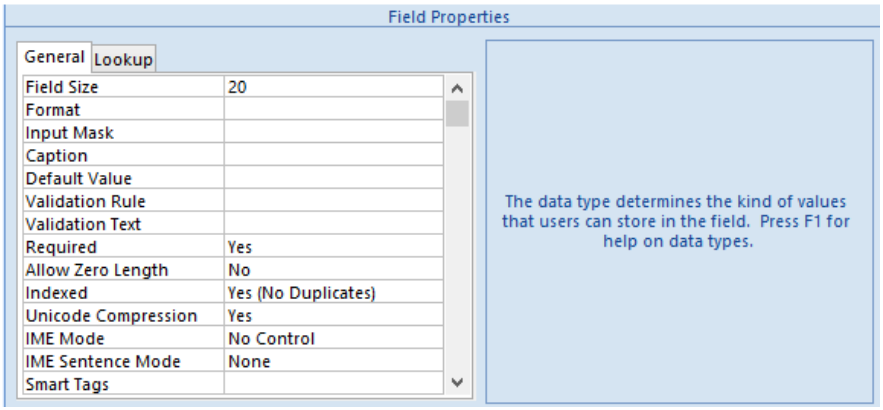
3) Setelah seluruh kolom didefinisikan, langkah selanjutnya adalah menetapkan kunci utama (*primary key*), yaitu dengan menyorot horisontal kolom yang dimaksud, kemudian klik ikon tombol yang ada di bagian atas.



Gambar 2.8: Proses penentuan *primary key*

4) Jika diperlukan, Anda dapat mengisi **Field Property** yang ada di bawahnya. Tampilan **Field Property** berbeda-beda untuk tiap kolom, tergantung pada tipe datanya. Berikut ini beberapa properti yang ada dalam Access:
   - **Field Size**, digunakan untuk menentukan lebar maksimum untuk data yang disimpan dalam suatu field. *Field size* diperuntukkan bagi tipe data **Text**, **Number** dan **AutoNumber**.
   - **Format**, digunakan untuk mengatur tampilan angka, tanggal, waktu dan teks yang ditampilkan di layar maupun printer. Anda dapat menggunakan salah satu format yang telah terdefinisi atau membuat sendiri menggunakan simbol-simbol format. Properti format berbeda-beda untuk setiap tipe data.
   - **Input Mask**, digunakan untuk menentukan tampilan pada saat data dimasukkan, juga digunakan untuk mengendalikan nilai yang dapat dimasukkan.
   - **Decimal Places**, untuk menentukan jumlah angka desimal yang ditentukan.
   - **Caption**, untuk menampilkan informasi yang berguna untuk user sebagai judul kolom atau laporan.

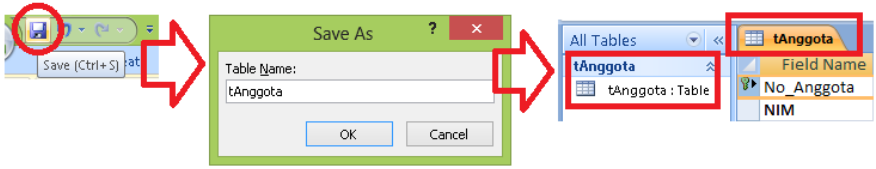
- **Default Value**, digunakan untuk menentukan nilai yang otomatis diisikan ke dalam suatu kolom ketika *record* baru diinputkan. Properti ini tidak berlaku untuk jenis data **AutoNumber** dan **OLE Object**.
- **Validation Rule**, digunakan untuk membatasi atau memasukkan data pada kolom tertentu.
- **Validation Text**, digunakan untuk menampilkan keterangan atau pesan apabila data yang dimasukkan tidak sesuai dengan batasan data yang telah ditentukan pada kolom tertentu.
- **Required**, digunakan untuk menentukan apakah sebuah kolom harus diisi atau tidak. Jika dipilih **Yes**, maka setiap kali mengisikan *record* baru, Anda harus mengisikan nilai ke dalam kolom ini. Jika isinya **No**, Anda boleh mengosongkannya.
- **Allow zero Length**, digunakan untuk menentukan validitas data dalam sebuah kolom. Bernilai **Yes**, jika data tersebut dianggap **VALID**, dan **No** jika jika dianggap **TIDAK VALID**. Properti ini hanya berlaku untuk tipe data **Text**, **Memo** dan **Hyperlink**.
- **Indexed**, digunakan untuk membuat index pada kolom tertentu.
- Dan berbagai atribut properti lainnya dapat Anda pelajari sendiri. Gambar 2.9 berikut ini merupakan tampilan **Field Properties**.



Gambar 2.9: Bagian **Field Properties**

5) Setelah selesai melakukan pendefinisian struktur tabel, simpanlah hasil pendefinisian tersebut dengan cara mengklik ikon tombol .

Setelah muncul kotak dialog **Save As**, masukkan nama tabel (misal "**tAnggota**") dan klik tombol **[OK]**.



Gambar 2.10: Proses simpan tabel "**tAnggota**"

6) Dengan cara yang sama, dapat dibuat tabel lain yang diperlukan dalam *database* yang bersangkutan. Berikut adalah tabel-tabel yang telah dibuat untuk aplikasi SIM Perpus.

tAnggota

Field Name	Data Type
No_Anggota	Number
NIM	Text
Nama	Text
Alamat	Text
Jenis_Kelamin	Number
Angkatan	Number
Kode_Prodi	Number

tDaftar_Buku

Field Name	Data Type
Kode_Buku	Number
ISBN	Text

Gambar 2.11: Struktur tabel **tAnggota** dan **tDaftar_Buku**

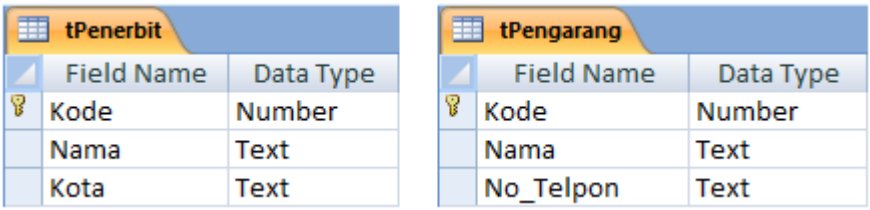
tInfo_Buku

Field Name	Data Type
ISBN	Text
DCC	Text
Judul	Text
Deskripsi	Text
Kode_Pengarang	Number
Kode_Penerbit	Number
Tahun_Terbit	Number

tJenis_Buku

Field Name	Data Type
Kode_DCC	Text
Jenis_Buku	Text

Gambar 2.12: Struktur tabel **tInfo_Buku** dan **tJenis_Buku**



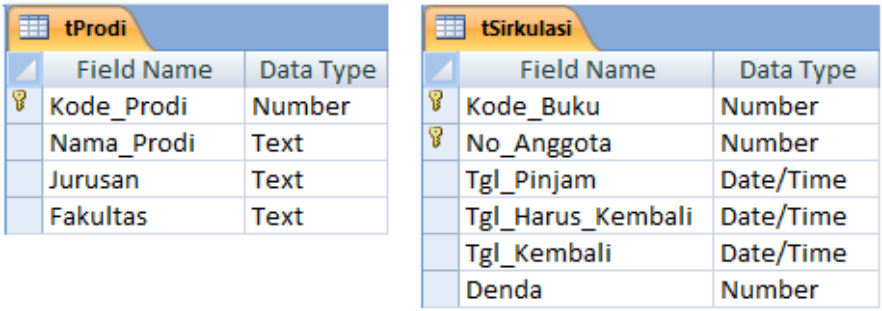
tPenerbit

Field Name	Data Type
Kode	Number
Nama	Text
Kota	Text

tPengarang

Field Name	Data Type
Kode	Number
Nama	Text
No_Telpon	Text

Gambar 2.13: Struktur tabel **tPenerbit** dan **tPengarang**



tProdi

Field Name	Data Type
Kode_Prodi	Number
Nama_Prodi	Text
Jurusan	Text
Fakultas	Text

tSirkulasi

Field Name	Data Type
Kode_Buku	Number
No_Anggota	Number
Tgl_Pinjam	Date/Time
Tgl_Harus_Kembali	Date/Time
Tgl_Kembali	Date/Time
Denda	Number

Gambar 2.14: Struktur tabel **tProdi** dan **tSirkulasi**

2.6 Membuat dan Mengatur Hubungan Antar Tabel

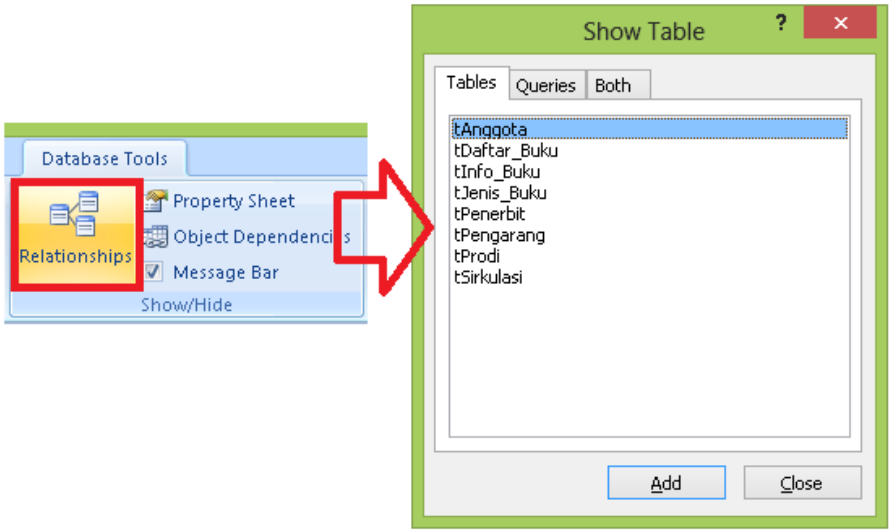
Pada saat merancang suatu sistem informasi yang memerlukan *database* yang besar, Anda mungkin perlu mengaitkan atau membuat hubungan antara suatu tabel dengan tabel lainnya. Dengan tujuan agar *database* yang dirancang bisa lebih efisien dan efektif.

2.6.1 Membuat Hubungan Antar Tabel

Hubungan antar tabel bekerja dengan mencocokkan data dalam kolom kunci, biasanya berupa kolom yang memiliki **tipe data** yang sama pada kedua tabel yang memiliki hubungan. Pada umumnya, kolom-kolom yang bersesuaian ini adalah **primary key** pada tabel yang satu, yang memberikan identitas unik bagi tiap *record* dalam tabel tersebut dan **foreign key** pada tabel yang lainnya. Untuk membuat hubungan antar tabel, langkahnya adalah sebagai berikut :

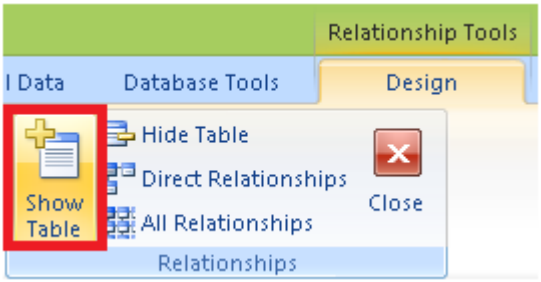
1) Pastikan file database yang tabelnya akan dihubungkan telah aktif, kemudian dari tab menu **Database Tools** klik ikon **Relationships**.
2) Setelah muncul kotak dialog **Show Table** akan ditampilkan, klik dan pilih nama tabel yang ingin dihubungkan. Kemudian klik tombol

[Add]. Untuk memilih beberapa tabel sekaligus, lakukan pemilihan sambil menekan tombol **[Ctrl]** pada *keyboard* Anda**.**



Gambar 2.15: Kotak dialog **Show Table**

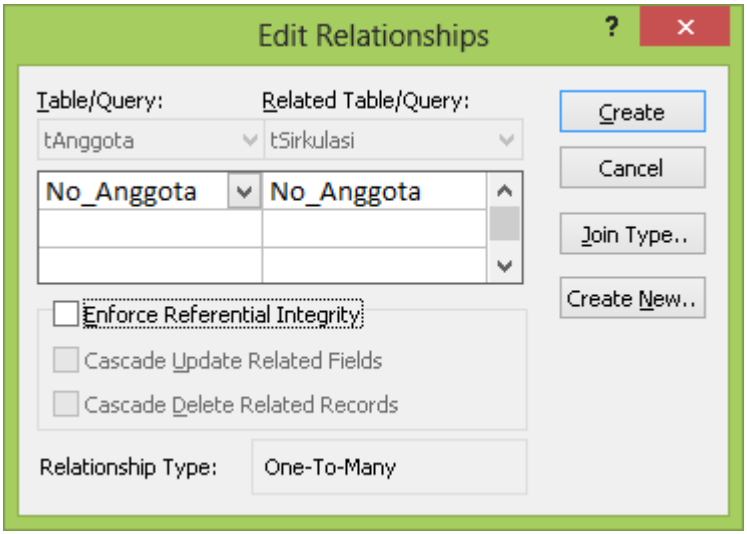
3) Klik tombol **[Close]**, untuk menutup kotak dialog **Show Table** diatas, sedangkan jika ingin kotak dialog ini tampil kembali, pilih menu tab **Relationships Tools** dan klik ikon tombol **[Show Table]**.



Gambar 2.16: Ikon tombol **[Show Table]**

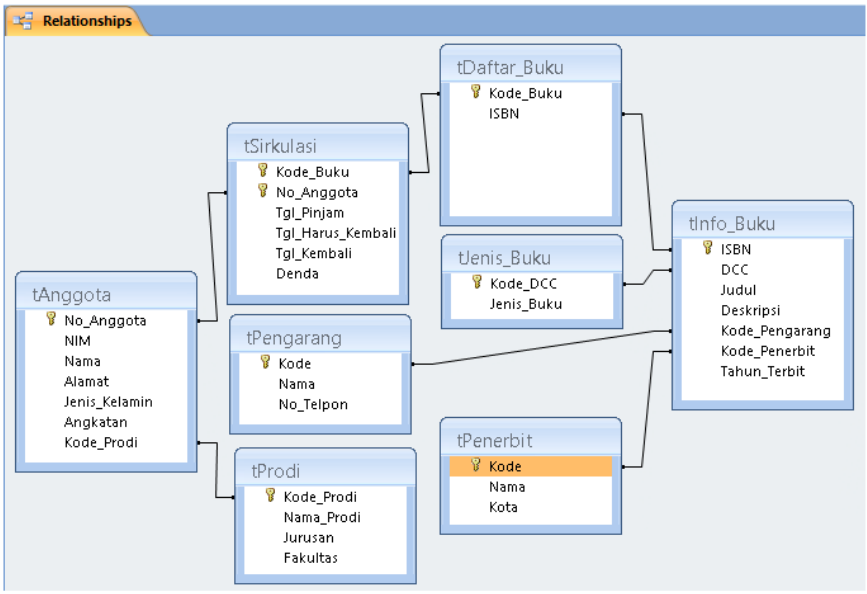
4) Untuk membuat hubungan antar tabel, Anda cukup tekan-geser (*drag*) nama kolom yang ingin gunakan sebagai kunci penghubung ke posisi nama kolom pada tabel yang diinginkan. Sebagai contoh

drag kolom **No_Anggota** yang berada pada tabel **tAnggota** ke posisi kolom **No_Anggota** yang berada pada tabel **tSirkulasi.**



Gambar 2.17: Kotak dialog **Edit Relationships**

5) Setelah muncul kotak dialog **Edit Relationships**, pastikan nama kolom yang akan dijadikan kunci penghubung sudah sesuai, klik tombol perintah **[Create]**. Untuk relasi antar tabel yang lain dapat dilakukan dengan langkah yang sama. Gambar 2.18 berikut ini merupakan tampilan hubungan antar-tabel yang terlibat dalam database **dbperpus.accdb**.



Gambar 2.18: Relasi antar tabel pada jendela **Relationships**

2.6.2 Mengatur Hubungan Antar Tabel

Seperti yang telah dijelaskan sebelumnya, hubungan antar tabel dihubungkan oleh kolom penghubung. Apabila diperlukan, Anda dapat mengatur hubungan antar tabel tersebut dengan menggunakan langkah sebagai berikut:

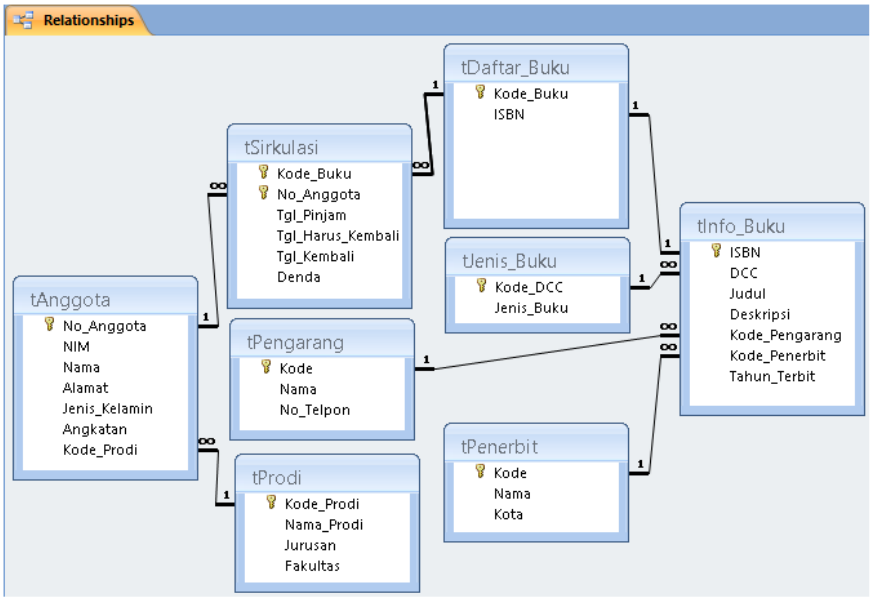
1. Pada jendela **Relationships** yang sedang ditampilkan, pilih dann klik dua kali (*double clicked*) **garis penghubung** antar tabel yang ingin diatur. Misalkan garis yang menghubungkan kolom **No_Anggota** pada tabel **tAnggota** dan kolom **No_Anggota** yang berada pada tabel **tSirkulasi.**
2. Setelah muncul kotak dialog **Edit Relationships**, Anda dapat mengubah kolom penghubung yang sekarang digunakan dengan kolom penghubung yang lain, kemudian klik tombol **[OK]**.

Hal-hal yang perlu diperhatikan dalam melakukan pengaturan hubungan antar tabel adalah sebagai berikut.

1) Apabila Anda memberi tanda **CENTANG** pada opsi **Enforce Referential Integrity,** maka tanda jenis hubungan antar tabel akan ditampilkan. Contoh hubungan **ONE-TO-MANY** (hubungan antara

tabel **tAnggota** dan **tSirkulasi**) ditunjukkan seperti Gambar 2.17 di atas.

2) Anda dapat melakukan dua pengecualian terhadap **REFERENTIAL INTEGRITY**, yakni dengan memilih atau klik pada opsi berikut ini :
   - **Cascade Update Related Fields**, perubahan *record* pada kolom **PRIMARY KEY** (**TABEL SUMBER**) secara otomatis akan mengubah nilai pada *record-record* yang bersesuaian pada kolom **FOREIGN KEY** (**TABEL TUJUAN** yang memiliki hubungan dengan **TABEL SUMBER**).
   - **Cascade Delete Related Records**, penghapusan *record* pada **TABEL SUMBER** secara otomatis akan menghapus *record-record* yang bersesuaian pada **TABEL TUJUAN**.
3) Gambar 2.19 berikut ini merupakan hubungan antar tabel yang menunjukkan jenis hubungan antar tabel penyusun aplikasi SIM Perpus.



Gambar 2.19: Jenis hubungan antar tabel pada jendela **Relationships**

2.7 Penggunaan SQL melalui Fitur Query

Access menawarkan metode fleksibel dalam pengambilan data yang memungkinkan Anda mencari informasi yang dibutuhkan untuk

menjawab pertanyaan tertentu. Filter memungkinkan Anda mengeluarkan data yang tidak relevan, memberi Anda tampilan yang jelas terhadap data yang diinginkan. Sebagaimana struktur tabel diatas, sebelum dibahas perintah SQL, berikut disajikan beberapa contoh data (*record*) pada masing-masing tabel yang akan digunakan uji coba penerapan perintah SQL.

tProdi

Kode_Prodi	Nama_Prodi	Jurusan	Fakultas
1	Administrasi Pendidikan	Administrasi Pendidikan	Fakultas Ilmu Pendidikan
2	Manajemen Pendidikan	Administrasi Pendidikan	Fakultas Ilmu Pendidikan
3	Bimbingan dan Konseling	Psikologi Pendidikan dan Bimbingan	Fakultas Ilmu Pendidikan

Gambar 2.20: Contoh data pada tabel **tProdi**

tAnggota

No_Anggota	NIM	Nama	Alamat	Jenis_Kelam	Angkatan	Kode_Prodi
10001	100123	Mariya Ulfa	Karangmalang A.50	0	2011	1
10002	100124	Sugiharti	Kuningan E.12	0	2012	1
10003	100125	Ahmad Sudiro	Sendowo F.23	1	2011	1
10004	200126	Sholihun	Karangwuni G.13	1	2012	2
10005	200128	Titan Sita	Karangmalang D.15	0	2013	2
10006	300127	Supirso	Blimbingsari F.31	1	2011	3

Gambar 2.21: Contoh data pada tabel **tAnggota**

tPenerbit

Kode	Nama	Kota
1011	Gava Media	Yogyakarta
1012	Graha Ilmu	Yogyakarta
1013	Kanisius	Sleman
1014	Andi Offset	Sleman
1015	Informatika	Bandung
1016	Elex Media Komputindo	Jakarta
1017	UNY Press	Sleman
1018	UGM Press	Sleman

Gambar 2.22: Contoh data pada tabel **tPenerbit**

tPengarang

Kode	Nama	No_Telpon
1101	Lantip Diat Prasojo	-
1102	Sudiyono	-
1103	Sugiyono	-
1104	Ngatio	-
1105	Riyanto	-

Gambar 2.23: Contoh data pada tabel **tPengarang**

tJenis_Buku

Kode_DCC	Jenis_Buku
1011	Teknologi Informasi
1012	Ilmu Pendidikan
1013	Bahasa dan Sastra
1014	Ilmu Sains
1015	Ekonomi dan Bisnis
1016	Kedokteran

Gambar 2.24: Contoh data pada tabel **tJenis_Buku**

tInfo_Buku

ISBN	DCC	Judul	Deskripsi	Kode_Pengarang	Kode_Penerbit	Tahun_Terb
111-2222-33-1	1011	Teknologi Informasi Pendidikan	-	1101	1011	2010
111-2222-33-2	1012	Supervisi Pendidikan	-	1101	1011	2011
222-3333-44-1	1012	Sistem Informasi Manajemen	-	1101	1017	2012
888-6666-77-1	1011	Pemrograman PHP-Ms. SQL Serve	-	1104	1013	2012
888-6666-77-2	1012	Penelitian Tindakan Kelas	-	1104	1015	2013
888-6666-77-3	1013	Puisi untuk Bangsaku	-	1104	1017	2014
999-7777-33-1	1011	Pemrograman Java SE/ME/EE	-	1105	1011	2009
999-7777-33-2	1011	Pemrograman PHP-MySQL	-	1105	1011	2011

Gambar 2.25: Contoh data pada tabel **tInfo_Buku**

tDaftar_Buku

Kode_Buku	ISBN
100001	111-2222-33-1
100002	111-2222-33-2
100003	222-3333-44-1
100004	888-6666-77-1
100005	888-6666-77-2
100006	888-6666-77-3
100007	999-7777-33-1
100008	999-7777-33-2

Gambar 2.26: Contoh data pada tabel **tDaftar_Buku**

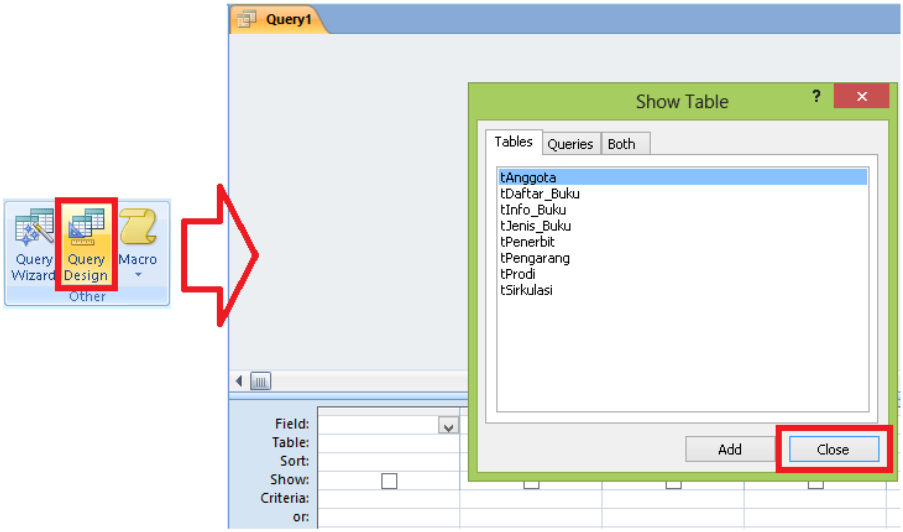
tSirkulasi

Kode_Buku	No_Anggota	Tgl_Pinjam	Tgl_Harus_Kembali	Tgl_Kembal	Denda
100001	10001	02/10/2014	05/10/2014	04/10/2014	0
100001	10002	11/10/2014	14/10/2014	13/10/2014	0
100002	10001	17/10/2014	20/10/2014	20/10/2014	0
100003	10002	16/10/2014	19/10/2014	17/10/2014	0
100004	10004	18/10/2014	21/10/2014	22/10/2014	2000
100006	10004	18/10/2014	21/10/2014	21/10/2014	0
100005	10005	21/10/2014	24/10/2014	23/10/2014	0
100001	10005	21/10/2014	24/10/2014	23/10/2014	0
100002	10006	20/10/2014	23/10/2014	23/10/2014	0
100007	10006	20/10/2014	23/10/2014	23/10/2014	0
100008	10006	19/10/2014	22/10/2014	23/10/2014	2000

Gambar 2.27: Contoh data pada tabel **tSirkulasi**

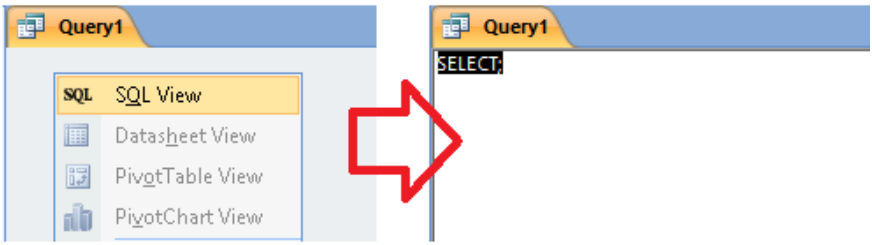
Untuk lebih memahami tentang perintah SQL, berikut ini disajikan beberapa contoh penerapan SQL pada satu tabel, *nested* SQL atau subquery, dan SQL pada multi tabel. Untuk memulai menggunakan perintah SQL, langkah awal yang perlu dilakukan adalah sebagai berikut:

1) Pada tab menu **Create**, klik ikon tombol **[Query Design]**.
2) Setelah muncul jendela **Query** (biasanya diberi nama "**Query1**") yang dibarengi kotak dialog **Show Table**, klik tombol **[Close]**.



Gambar 2.28: Jendela **Query** dan kotak dialog **Show Table**

3) Pada jendela **Query**, klik kanan dan pilih menu **SQL VIEW** sehingga jendela **Query** berubah menjadi **SQL EDITOR** dengan perintah awal yang muncul, yaitu "**SELECT**". Melalui jendela **Query-SQL EDITOR** inilah perintah SQL diketik dan dieksekusi.

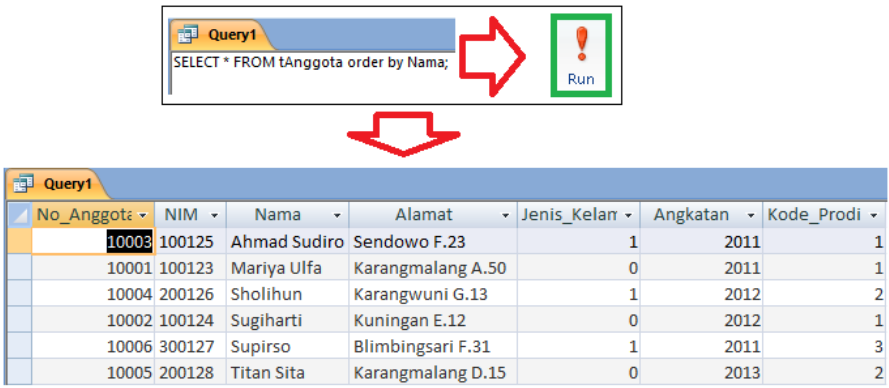


Gambar 2.29: Jendela **Query-SQL EDITOR**

2.7.1 Penerapan SQL pada Satu Tabel

1) Menampilkan seluruh data yang ada pada tabel **tAnggota** yang diurutkan berdasarkan kolom **Nama**.

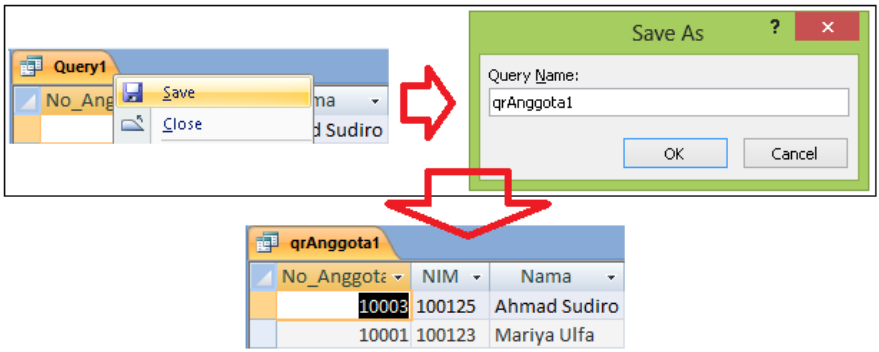
```
SELECT *
FROM tAnggota
ORDER BY Nama;
```



No_Anggota	NIM	Nama	Alamat	Jenis_Kelam	Angkatan	Kode_Prodi
10003	100125	Ahmad Sudiro	Sendowo F.23	1	2011	1
10001	100123	Mariya Ulfa	Karangmalang A.50	0	2011	1
10004	200126	Sholihun	Karangwuni G.13	1	2012	2
10002	100124	Sugiharti	Kuningan E.12	0	2012	1
10006	300127	Supirso	Blimbingsari F.31	1	2011	3
10005	200128	Titan Sita	Karangmalang D.15	0	2013	2

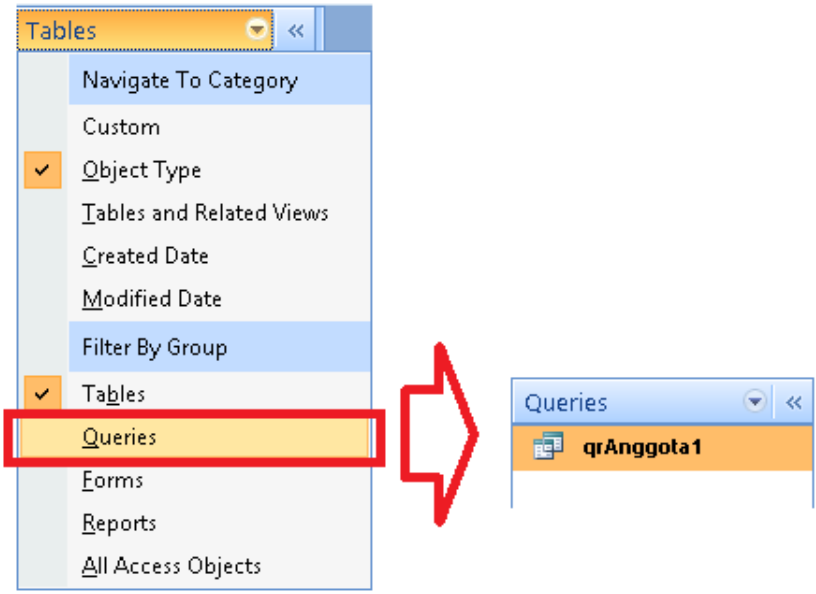
Gambar 2.30: Proses eksekusi SQL pada Access

Untuk menghindari penulisan ulang kode SQL yang telah diketik, Anda dapat menyimpan perintah SQL tersebut dalam bentuk **Query**. Caranya, cukup klik kanan bagian *header* jendela **Query** dan pilih menu **Save**. Setelah muncul kotak dialog **Save As**, masukkan nama **Query** yang diinginkan (misal "**qrAnggota1**") dan klik tombol **[OK]**.



Gambar 2.31: Proses simpan SQL dalam bentuk **Query** pada Access

Jika tidak terjadi kesalahan, maka akan muncul nama **Query** yang disimpan dalam daftar komponen **Query**.



Gambar 2.32: Menampilkan daftar **Query** yang tersimpan

2) Menampilkan data **Nama**, **NIM** dan **Jenis Kelamin** yang ada pada tabel **tAnggota** dengan mengganti nilai **1** (satu) menjadi "**Laki-laki**" dan **0** (nol) menjadi "**Perempuan**", dan diurutkan berdasarkan kolom **Nama**, kemudian simpan dengan nama "**qrAnggotaJK**".

```
SELECT Nama, NIM, iif(Jenis_Kelamin=1, 'Laki-laki',
'Perempuan') AS JK
FROM tAnggota
ORDER BY Nama;
```

Output:

qrAnggotaJK

Nama	NIM	JK
Ahmad Sudiro	100125	Laki-laki
Mariya Ulfa	100123	Perempuan
Sholihun	200126	Laki-laki
Sugiharti	100124	Perempuan
Supirso	300127	Laki-laki
Titan Sita	200128	Perempuan

Gambar 2.33: Hasil eksekusi *query* **qrAnggotaJK**

3) Menampilkan data **Nama**, **NIM**, **Angkatan** dan **Alamat** yang ada pada tabel **tAnggota** yang memenuhi kondisi **Tahun Angkatan Ganjil** dan diurutkan berdasarkan kolom **Nama** dan **Angkatan**, kemudian simpan dengan nama "**qrAnggotaThnGanjil**".

```
SELECT Nama, NIM,  Angkatan, Alamat
FROM tAnggota
Where (Angkatan mod 2 = 1)
ORDER BY Nama, Angkatan;
```

Output:

qrAnggotaThnGanjil

Nama	NIM	Angkatan	Alamat
Ahmad Sudiro	100125	2011	Sendowo F.23
Mariya Ulfa	100123	2011	Karangmalang A.50
Supirso	300127	2011	Blimbingsari F.31
Titan Sita	200128	2013	Karangmalang D.15

Gambar 2.34: Hasil eksekusi *query* **qrAnggotaThnGanjil**

4) Menampilkan jumlah anggota yang dikelompokkan berdasarkan kolom **Jenis_Kelamin**, kemudian simpan dengan nama "**qrAnggotaRekapJK**".

```
SELECT iif(Jenis_Kelamin=1, 'Laki-laki', 'Perempuan')
AS JK, COUNT(*) as Total
FROM tAnggota
GROUP BY Jenis_Kelamin;
```

Output:

qrAnggotaRekapJK

JK	Total
Perempuan	3
Laki-laki	3

Gambar 2.35: Hasil eksekusi *query* **qrAnggotaRekapJK**

5) Menampilkan data **Kota** yang ada di dalam tabel **Penerbit** yang jumlahnya lebih dari 1 (satu),kemudian simpan dengan nama "**qrPenerbitRekapKota**" .

```
SELECT Kota, Count(*) AS Total
```

```
FROM tPenerbit
GROUP BY Kota
Having Count(*)>1;
```

Output:

qrPenerbitRekapKota

Kota	Total
Sleman	4
Yogyakarta	2

Gambar 2.36: Hasil eksekusi *query* **qrPenerbitRekapKota**

6) Menampilkan data **Kode**, **Nama** dan **Kota** yang ada di dalam tabel **tPenerbit** yang memenuhi kondisi dalam kota tersebut (**Yogyakarta** dan **Sleman**) terdapat lebih dari satu penerbit, kemudian simpan dengan nama "**qrPenerbitKota1**".

```
SELECT *
FROM tPenerbit
Where Kota in ('Yogyakarta', 'Sleman');
```

Output:

qrPenerbitKota1

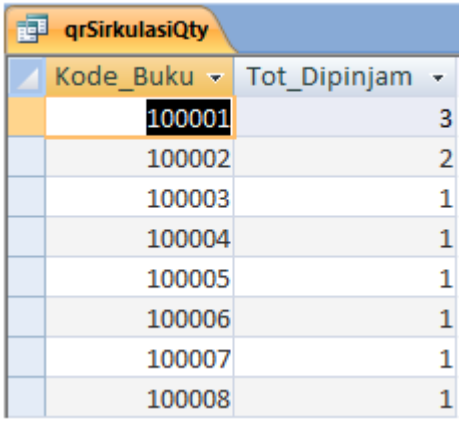
Kode	Nama	Kota
1011	Gava Media	Yogyakarta
1012	Graha Ilmu	Yogyakarta
1013	Kanisius	Sleman
1014	Andi Offset	Sleman
1017	UNY Press	Sleman
1018	UGM Press	Sleman

Gambar 2.37: Hasil eksekusi *query* **qrPenerbitKota1**

7) Menampilkan data **Kode_Buku** dan total berapa kali dipinjam dalam tabel **tSirkulasi**, kemudian simpan dengan nama "**qrSirkulasiQty**".

```
SELECT Kode_Buku, Count(*) as Tot_Dipinjam
FROM tSirkulasi
GROUP BY Kode_Buku;
```

Output:



qrSirkulasiQty

Kode_Buku	Tot_Dipinjam
100001	3
100002	2
100003	1
100004	1
100005	1
100006	1
100007	1
100008	1

Gambar 2.38: Hasil eksekusi *query* **qrSirkulasiQty**

8) Menampilkan seluruh data pada tabel **tInfo_Buku** yang judulnya mengandung kata "pendidikan", kemudian simpan dengan nama "**qrInfoBukuFilter**".

```
SELECT *
FROM tInfo_buku
WHERE judul LIKE "*pendidikan*";
```

Output:

qrInfoBukuFilter

ISBN	DCC	Judul	Deskripsi	Kode_Pengarang	Kode_Penerbit	Tahun_Terb
111-2222-33-1	1011	Teknologi Informasi Pendidikan	-	1101	1011	2010
111-2222-33-2	1012	Supervisi Pendidikan	-	1101	1011	2011

Gambar 2.39: Hasil eksekusi *query* **qrInfoBukuFilter**

9) Menampilkan 40 persen teratas dari seluruh informasi yang ada pada tabel **tInfo_Buku**.

```
SELECT TOP 40 PERCENT *
FROM tInfo_Buku;
```

Output:

qrInfoBukuTop40

ISBN	DCC	Judul	Deskripsi	Kode_Pengarang	Kode_Penerbit	Tahun_Terb
111-2222-33-1	1011	Teknologi Informasi Pendidikan	-	1101	1011	2010
111-2222-33-2	1012	Supervisi Pendidikan	-	1101	1011	2011
222-3333-44-1	1012	Sistem Informasi Manajemen	-	1101	1017	2012
999-7777-33-1	1011	Pemrograman Java SE/ME/EE	-	1105	1011	2009

Gambar 2.40: Hasil eksekusi *query* **qrInfoBukuTop40**

10) Menyimpan seluruh isi tabel **tInfo_Buku** dengan nama yang lain, biasanya digunakan untuk membuat *temporary table*, kemudian simpan dengan nama "**qrTempInfoBuku**".

```
SELECT *
FROM tInfo_Buku AS Temp_InfoBuku;
```

Output:

qrTempInfoBuku

ISBN	DCC	Judul	Deskripsi	Kode_Pengarang	Kode_Penerbit	Tahun_Terb
111-2222-33-1	1011	Teknologi Informasi Pendidikan	-	1101	1011	2010
111-2222-33-2	1012	Supervisi Pendidikan	-	1101	1011	2011
222-3333-44-1	1012	Sistem Informasi Manajemen	-	1101	1017	2012
888-6666-77-1	1011	Pemrograman PHP-Ms. SQL Serve	-	1104	1013	2012
888-6666-77-2	1012	Penelitian Tindakan Kelas	-	1104	1015	2013
888-6666-77-3	1013	Puisi untuk Bangsaku	-	1104	1017	2014
999-7777-33-1	1011	Pemrograman Java SE/ME/EE	-	1105	1011	2009
999-7777-33-2	1011	Pemrograman PHP-MySQL	-	1105	1011	2011

Gambar 2.41: Hasil eksekusi *query* **qrTempInfoBuku**

2.7.2 Penerapan Nested SQL (Sub-Query)

1) Menampilkan data **Kode**, **Nama** dan **Kota** yang ada di dalam tabel **tPenerbit** yang memenuhi kondisi dalam kota tersebut (**Yogyakarta** dan **Sleman**) terdapat lebih dari satu penerbit, kemudian simpan dengan nama "**qrPenerbitKota2**".

```
SELECT Kode, Nama, Kota
FROM tPenerbit
WHERE Kota in (SELECT Kota FROM tPenerbit GROUP BY
Kota HAVING Count(*)>1);
```

Output:

qrPenerbitKota2

Kode	Nama	Kota
1011	Gava Media	Yogyakarta
1012	Graha Ilmu	Yogyakarta
1013	Kanisius	Sleman
1014	Andi Offset	Sleman
1017	UNY Press	Sleman
1018	UGM Press	Sleman

Gambar 2.42: Hasil eksekusi *query* **qrPenerbitKota2**

2) Menampilkan informasi buku terbaru (dalam hal ini diwakili oleh **Judul** dan **Tahun Terbit**), dengan membandingkan **tahun terbit**

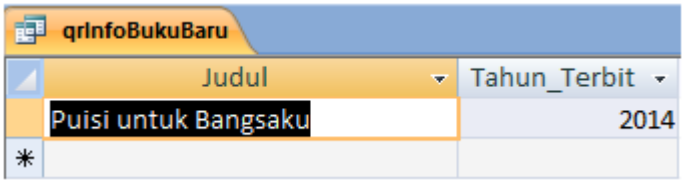
dengan semua **tahun terbit** yang lain, tahun terbit yang paling besar yang ditampilkan, kemudian simpan dengan nama "**qrInfoBukuBaru**".

```
SELECT Judul, Tahun_Terbit
FROM tInfo_Buku
WHERE   Tahun_terbit>=all(SELECT   Tahun_Terbit   FROM
tInfo_Buku);
```

Atau

```
SELECT Judul, Tahun_Terbit
FROM tInfo_Buku
WHERE   Tahun_terbit>=(SELECT   MAX(Tahun_Terbit)   FROM
tInfo_Buku);
```

Output:



Gambar 2.43: Hasil eksekusi *query* **qrInfoBukuBaru**

3) Menampilkan data **Judul** dan **Tahun Terbit**, dengan membandingkan **Tahun Terbit** buku tertentu dengan semua **Tahun Terbit** yang lain, yang ditampilkan **SELAIN TAHUN TERBIT TERKECIL** (selain **2009**), kemudian simpan dengan nama "**qrInfoBukuMoreANY**".

```
SELECT Judul, Tahun_Terbit
FROM tInfo_Buku
WHERE   Tahun_Terbit>ANY(SELECT   Tahun_terbit   FROM
tInfo_Buku);
```

Output:

qrInfoBukuMoreANY

Judul	Tahun_Terb
Teknologi Informasi Pendidikan	2010
Supervisi Pendidikan	2011
Sistem Informasi Manajemen	2012
Pemrograman PHP-MySQL	2011
Pemrograman PHP-Ms. SQL Serve	2012
Penelitian Tindakan Kelas	2013
Puisi untuk Bangsaku	2014

Gambar 2.44: Hasil eksekusi *query* **qrInfoBukuMoreANY**
(Menampilkan data **SELAIN TAHUN TERBIT TERKECIL)**

4) Menampilkan data **Judul** dan **tahun Terbit**, dengan membandingkan **tahun terbit** buku tertentu dengan semua **tahun terbit** yang lain, yang ditampilkan **SELAIN TAHUN TERBIT TERBESAR** (selain **2014**), kemudian simpan dengan nama "**qrInfoBukuLessANY**".

```
SELECT Judul, Tahun_Terbit
FROM tInfo_Buku
WHERE    Tahun_Terbit<ANY(SELECT    Tahun_terbit    FROM
tInfo_Buku);
```

Output:

qrInfoBukuLessANY

Judul	Tahun_Terb
Teknologi Informasi Pendidikan	2010
Supervisi Pendidikan	2011
Sistem Informasi Manajemen	2012
Pemrograman Java SE/ME/EE	2009
Pemrograman PHP-MySQL	2011
Pemrograman PHP-Ms. SQL Serve	2012
Penelitian Tindakan Kelas	2013

Gambar 2.45: Hasil eksekusi *query* **qrInfoBukuLessANY**
(Menampilkan data **SELAIN TAHUN TERBIT TERBESAR)**

5) Menampilkan data **Kode** dan **Nama** dari tabel **tPengarang** yang kode pengarang tersebut ada di dalam tabel **tInfo_Buku**, kemudian simpan dengan nama "**qrPengarangExist**".

```
SELECT Kode, Nama
FROM tPengarang
WHERE EXISTS(SELECT * FROM tInfo_Buku WHERE Kode =
tInfo_Buku.Kode);
```

Atau

```
SELECT Kode, Nama
FROM tPengarang
WHERE Kode IN (SELECT Kode_Pengarang FROM tInfo_Buku);
```

Output:

qrPengarangExist

Kode	Nama
1101	Lantip Diat Prasojo
1104	Ngatio
1105	Riyanto

Gambar 2.46: Hasil eksekusi *query* **qrPengarangExist**

2.7.3 Penerapan SQL pada Multi Tabel

1) Menampilkan informasi Judul beserta Penerbitnya yang berasal dari penggabungan dua tabel (**tPenerbit** dan **tInfo_Buku**) dengan kondisi data **Kode_Penerbit** dari kedua tabel tadi sama.

```
SELECT a.ISBN, a.Judul, b.Nama as Penerbit
FROM tPenerbit b INNER JOIN tInfo_Buku a
ON b.Kode = a.Kode_Penerbit;
```

Output:

qrInnerBukuPenerbit

ISBN	Judul	Penerbit
111-2222-33-1	Teknologi Informasi Pendidikan	Gava Media
111-2222-33-2	Supervisi Pendidikan	Gava Media
999-7777-33-1	Pemrograman Java SE/ME/EE	Gava Media
999-7777-33-2	Pemrograman PHP-MySQL	Gava Media
888-6666-77-1	Pemrograman PHP-Ms. SQL Serve	Kanisius
888-6666-77-2	Penelitian Tindakan Kelas	Informatika
222-3333-44-1	Sistem Informasi Manajemen	UNY Press
888-6666-77-3	Puisi untuk Bangsaku	UNY Press

Gambar 2.47: Hasil eksekusi *query* **qrInnerBukuPenerbit**

2) Menampilkan data **Judul**, **ISBN**, **Pengarang**, **Penerbit**, dan **Tahun Terbit** yang berasal dari penggabungan tiga tabel (**tPenerbit**, **tPengarang** dan **tInfo_Buku**) dengan kondisi data pada kolom **tPenerbit.Kode** sama dengan data pada kolom **tInfo_Buku**.**Kode_Penerbit**, dan kolom **tInfo_Buku.Kode_Pengarang** sama dengan pada kolom **tPengarang.Kode**, kemudian simpan dengan nama "**qrInnerBukuPengarangPenerbit**".

```
SELECT  a.Judul, a.ISBN, b.Nama as Pengarang, c.Nama
as Penerbit, a.Tahun_Terbit
FROM tPengarang b INNER JOIN (tPenerbit c INNER JOIN
tInfo_buku a ON c.Kode = a.Kode_Penerbit) ON b.Kode =
a.Kode_Pengarang
ORDER BY b.Nama;
```

Output:

qrInnerBukuPengarangPenerbit

Judul	ISBN	Pengarang	Penerbit	Tahun_Terb
Sistem Informasi Manajemen	222-3333-44-1	Lantip Diat Prasojo	UNY Press	2012
Supervisi Pendidikan	111-2222-33-2	Lantip Diat Prasojo	Gava Media	2011
Teknologi Informasi Pendidikan	111-2222-33-1	Lantip Diat Prasojo	Gava Media	2010
Puisi untuk Bangsaku	888-6666-77-3	Ngatio	UNY Press	2014
Penelitian Tindakan Kelas	888-6666-77-2	Ngatio	Informatika	2013
Pemrograman PHP-Ms. SQL Serve	888-6666-77-1	Ngatio	Kanisius	2012
Pemrograman PHP-MySQL	999-7777-33-2	Riyanto	Gava Media	2011
Pemrograman Java SE/ME/EE	999-7777-33-1	Riyanto	Gava Media	2009

Gambar 2.48: Hasil eksekusi *query* **qrInnerBukuPengarangPenerbit**

3) Menampilkan data **Judul**, **ISBN** dan **Penerbit** yang berasal dari penggabungan dua tabel (tabel **tPenerbit** dan **tInfo_Buku**) dengan metode **LEFT JOIN**. Metode ini akan menampilkan seluruh data yang ada pada **LEFT TABLE** (tabel **tPenerbit**) meskipun **TIDAK ADA** di **RIGHT TABLE** (tabel **tInfo_Buku**). Simpan *query* ini dengan nama "**qrBukuPenerbitLeftJoin**".

```
SELECT a.Judul, a.ISBN, b.Nama as Penerbit
FROM tPenerbit b
LEFT JOIN tInfo_Buku a ON b.Kode=a.Kode_Penerbit
ORDER BY a.Judul
```

Output:

qrBukuPenerbitLeftJoin

Judul	ISBN	Penerbit
		UGM Press
		Elex Media Komputindo
		Andi Offset
		Graha Ilmu
Pemrograman Java SE/ME/EE	999-7777-33-1	Gava Media
Pemrograman PHP-Ms. SQL Serve	888-6666-77-1	Kanisius
Pemrograman PHP-MySQL	999-7777-33-2	Gava Media
Penelitian Tindakan Kelas	888-6666-77-2	Informatika
Puisi untuk Bangsaku	888-6666-77-3	UNY Press
Sistem Informasi Manajemen	222-3333-44-1	UNY Press
Supervisi Pendidikan	111-2222-33-2	Gava Media
Teknologi Informasi Pendidikan	111-2222-33-1	Gava Media

Gambar 2.49: Hasil eksekusi *query* **qrBukuPenerbitLeftJoin**

4) Menampilkan data **Judul**, **ISBN** dan **Penerbit** yang berasal dari penggabungan dua tabel (tabel **tPenerbit** dan **tInfo_Buku**) dengan metode **RIGHT JOIN**. Metode ini akan menampilkan seluruh data yang ada pada **RIGHT TABLE** (tabel **tInfo_Buku**) meskipun **TIDAK ADA** di **LEFT TABLE** (tabel **tPenerbit**). Simpan *query* ini dengan nama "**qrBukuPenerbitRightJoin**".

```
SELECT a.Judul, a.ISBN, b.Nama as Penerbit
FROM tPenerbit b
RIGHT JOIN tInfo_Buku a ON b.Kode=a.Kode_Penerbit
ORDER BY a.Judul
```

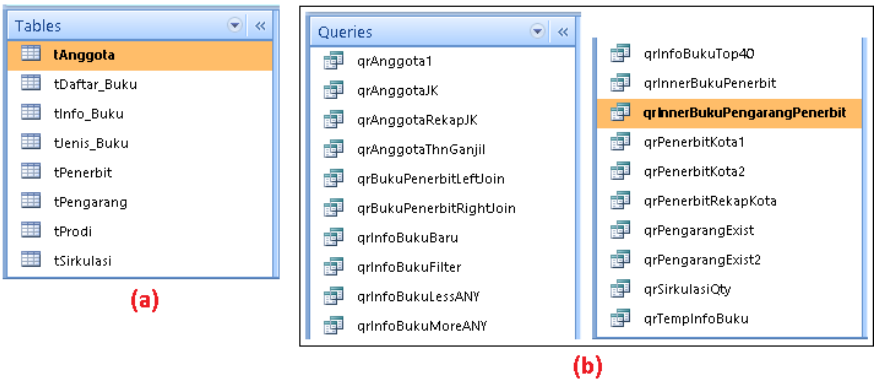
Output:

qrBukuPenerbitRightJoin

Judul	ISBN	Penerbit
Pemrograman Java SE/ME/EE	999-7777-33-1	Gava Media
Pemrograman PHP-Ms. SQL Server	888-6666-77-1	Kanisius
Pemrograman PHP-MySQL	999-7777-33-2	Gava Media
Penelitian Tindakan Kelas	888-6666-77-2	Informatika
Puisi untuk Bangsaku	888-6666-77-3	UNY Press
Sistem Informasi Manajemen	222-3333-44-1	UNY Press
Supervisi Pendidikan	111-2222-33-2	Gava Media
Teknologi Informasi Pendidikan	111-2222-33-1	Gava Media

Gambar 2.50: Hasil eksekusi *query* **qrBukuPenerbitRightJoin**

Gambar 2.51 berikut ini merupakan daftar tabel dan *query* yang telah dibuat. Anda tidak perlu mengikuti materi dengan mengetik ulang satu per satu, Anda dapat memanfaatkan file latihan yang ada di dalam CD Penyerta buku ini, tepatnya di dalam file "**dbperpus.accdb**".



Gambar 2.51: Daftar (a) tabel dan (b) *query* yang telah dibuat

3 Membuat Form dan Laporan dengan Microsoft Access

3.1 Pendahuluan

Dalam Access, form digunakan untuk memudahkan pengguna dalam melakukan pengolahan data, seperti: penyisipan (*insert*), pengubahan atau perbaharuan (*update*), penghapusan (*delete*) data, dan mengontrol alur aplikasi (*application flow*). Sama dengan tabel dan query, terdapat beberapa cara untuk membuat form, di antaranya yaitu :

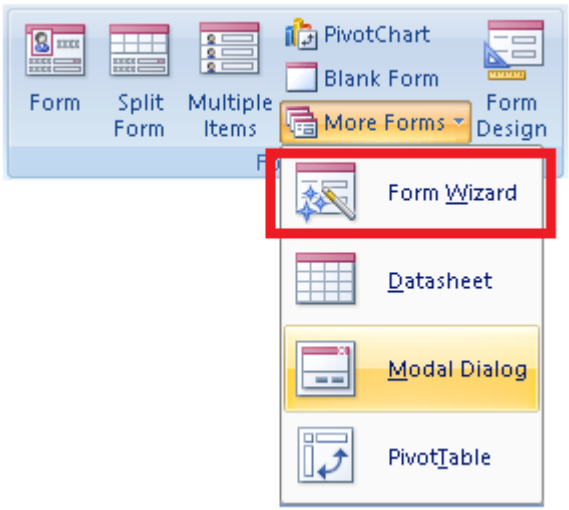
a. **Create form in Design view**, memungkinkan membuat form sesuai keinginan, yaitu dengan memasang beberapa komponen yang diinginkan secara manual sesuai kebutuhan. Cara ini relatif rumit dan memerlukan banyak proses.
b. **Create form by using wizard**, memungkinkan membuat form dengan cepat. Dengan cara ini, Anda cukup melakukan pemilihan nama tabel yang ingin akan diolah menggunakan form, kemudian memilih kolom mana saja yang perlu ditampilkan pada form.

Report dalam Access digunakan untuk membuat laporan. Laporan merupakan representasi dari hasil pengolahan data dari tabel dan/ query yang telah dibuat. Sama dengan form, report juga bisa dibuat dalam 2 cara, yaitu **Design View** dan **Wizard**. Kita akan membuat laporan dengan wizard. Misalkan laporan yang diinginkan adalah daftar anggota perpustakaan.

3.2 Membuat Form

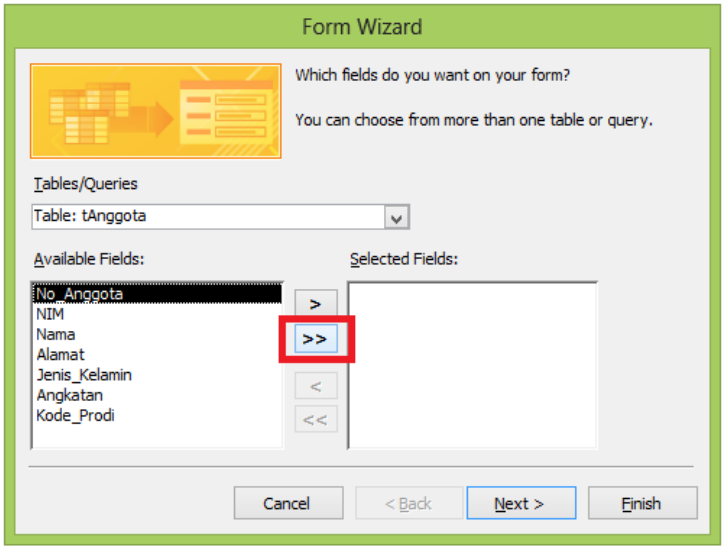
Dalam hal ini, akan dibuat form menggunakan cara **wizard**. Misalkan form untuk anggota perpustakaan. Berikut adalah langkah pembuatan form pada Microsoft Access:

1) Akses menu tab **Create**, pilih ikon tombol **[More Forms]**, pilih menu tombol **[Form Wizard]**.



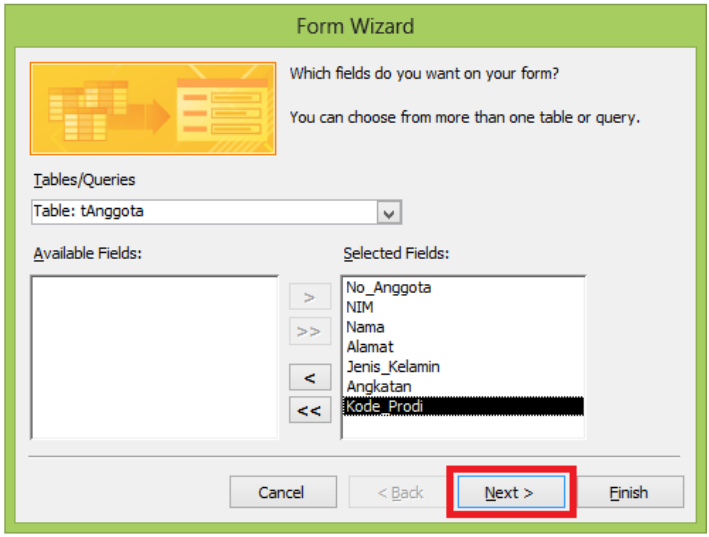
Gambar 3.1: Menu untuk membuat form

2) Setelah tampil kotak dialog pemilihan kolom dari tabel (atau queries) yang diinginkan. Tabel atau queries bisa dipilih lebih dari satu.



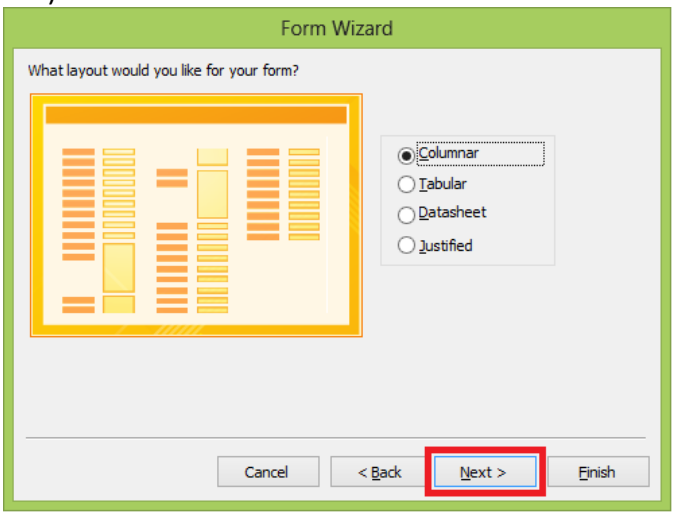
Gambar 3.2: Kotak dialog pemilihan field

3) Pilih **tAnggota** dan pilih kolom-kolom yang ingin ditampilkan pada form. Jika dipilih semua, klik tombol >>.



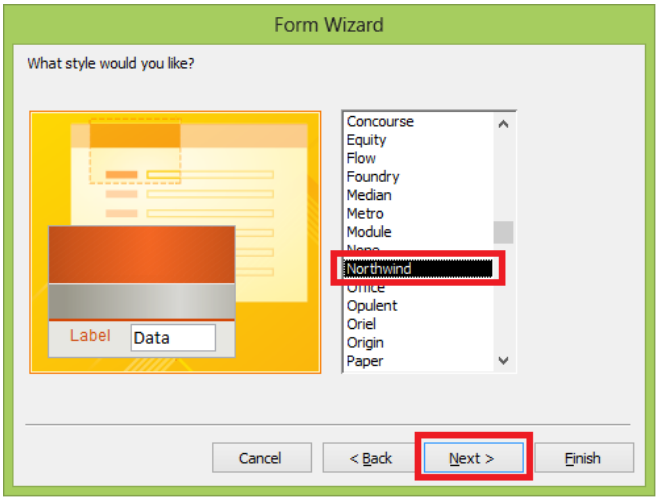
Gambar 3.3: Kotak dialog pemilihan field (field sudah dipilih)

4) Klik tombol **[Next]**, maka akan ditampilkan kotak dialog layout form, pilih opsi **Columnar** (untuk opsi yang lain bisa dicoba sendiri).



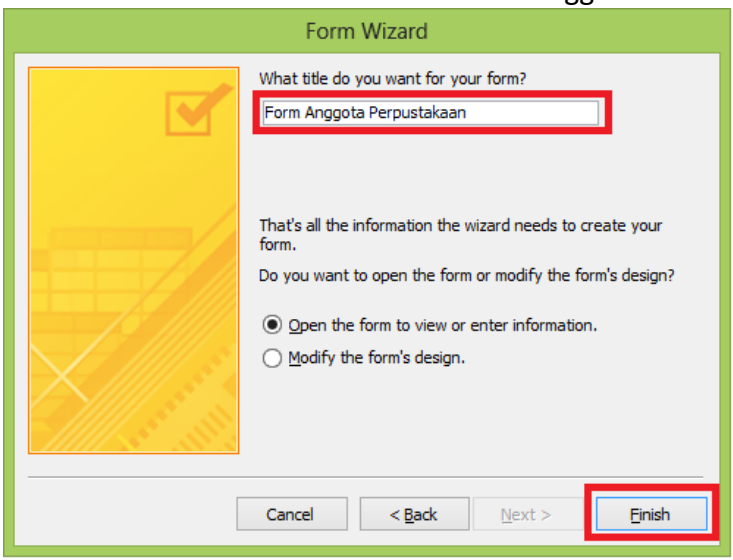
Gambar 3.4: Kotak dialog layout form

5) Klik tombol **[Next]**, maka akan ditampilkan kotak dialog style form dan pilih salah satu style, misalkan **International**.



Gambar 3.5: Kotak dialog style form

6) Klik tombol **[Next]**, maka akan ditampilkan kotak dialog title form. Misalkan kita masukkan Form Data Anggota.



Gambar 3.6: Kotak dialog title form

7) Setelah tombol **[Finish]** diklik, maka akan ditampilkan **Form Anggota Perpustakaan** seperti Gambar 3.7 berikut ini.

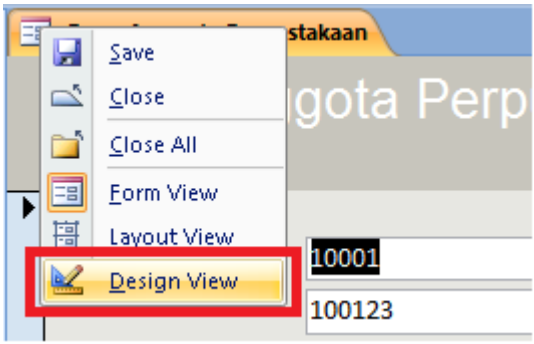
Gambar 3.7: Form Anggota Perpustakaan

Jika diperhatikan, pada kolom **Kode_Prodi**, pengguna mungkin masih kesulitan untuk mengisinya, karena harus mengingat **Kode Prodi** Anggota yang bersangkutan. Kolom **Kode_Prodi** dapat diubah sedemikian rupa agar Form Data Anggota lebih nyaman dioperasikan (*user friendly*).

3.2.1 Mengubah Text Box Menjadi Combo Box

Dalam hal ini akan dirapikan tampilannya dan juga kolom **Kode_Prodi** akan diubah menjadi **Combo Box** yang berisi data dari tabel **Prodi**. Berikut adalah langkah-langkahnya :

1) Klik kanan bagian *header* **Form Anggota Perpustakaan** dan pilih menu **Design View**.

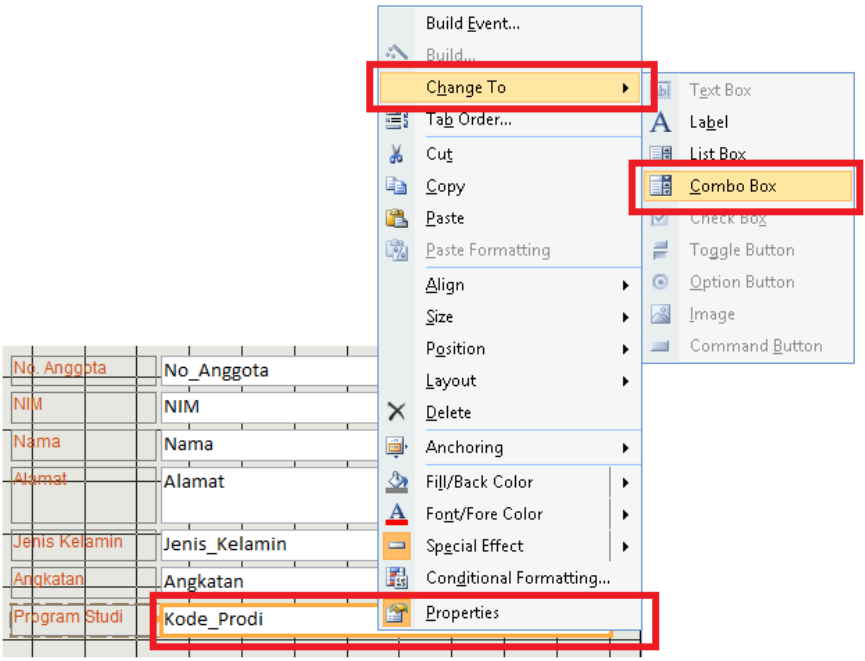


Gambar 3.8: Mengakses Form dalam format *design view*

2) Setelah tampilan form dalam format *desaign view*, ubah *caption* **No_Anggota** (menjadi **No. Anggota**), **Jenis_Kelamin** (menjadi **Jenis Kelamin**), dan **Kode_Prodi** (menjadi **Program Studi**).

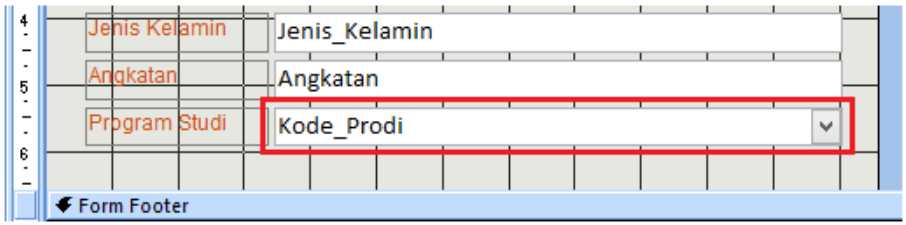
Gambar 3.9: Perubahan *caption* Form Anggota Perpustakaan

3) Klik kanan komponen *text box* **Kode_Prodi**, kemudian pilih **Change To > Combo Box**.



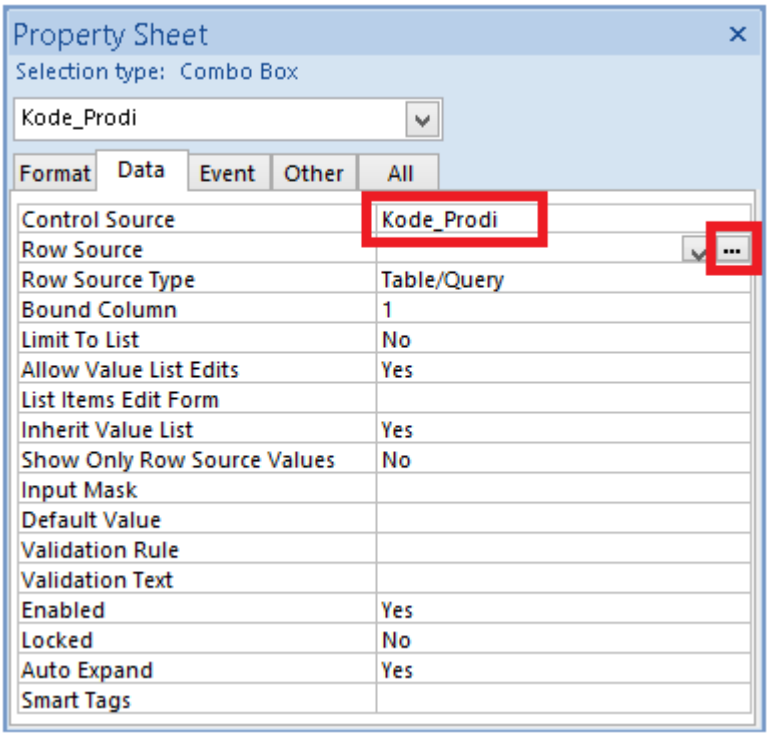
Gambar 3.10: Menu **Change To | Combo Box**

4) Gambar 3.11 menunjukkan hasil perubahan komponen *text box* **Kode_Prodi** menjadi komponen *combo box*.



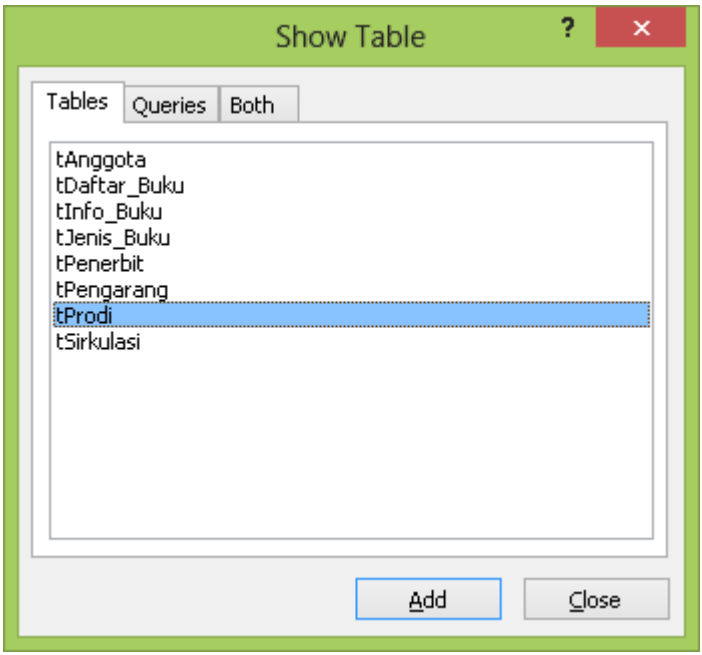
Gambar 3.11: Perubahan menjadi Combo Box

5) Langkah selanjutnya adalah melakukan klik kanan ke komponen **Kode_Prodi | Properties**. Maka akan ditampilkan Jendela Properti Combo Box seperti Gambar 3.12 berikut ini.



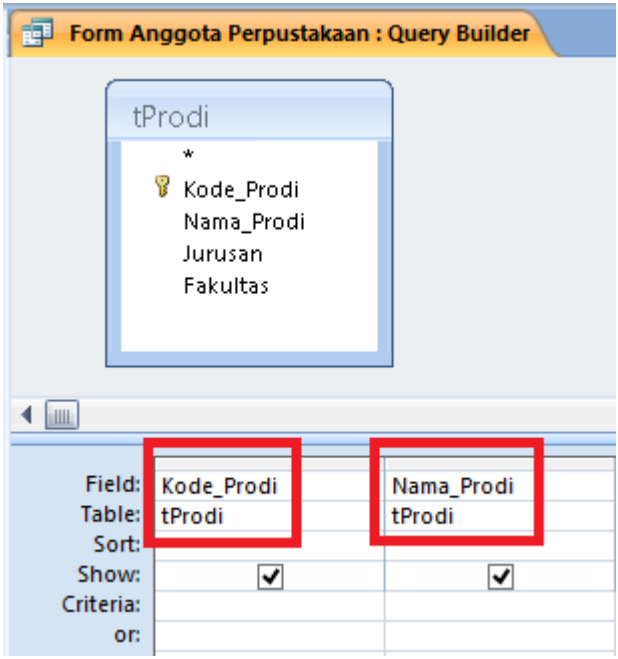
Gambar 3.12: Properti komponen *combo box* **Kode_Prodi**

6) Masukkan nilai "**Kode_Prodi**" untuk kolom **Control Source**, "**2**" (dua) untuk kolom **Column Count** dan klik kolom **Row Source**, maka akan ditampilkan kotak dialog **Show Table**.



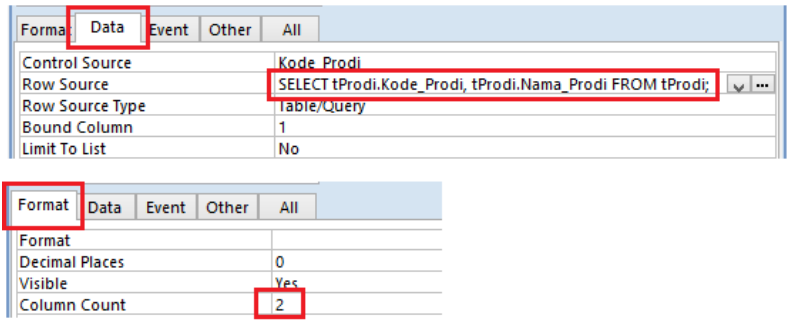
Gambar 3.13: Kotak dialog **Show Table**

7) Pilih tabel **tProdi**, klik tombol **[Add]**, kemudian klik tombol **[Close]**. Setelah tampil jendela **Query Builder**, lakukan langkah seperti yang ditunjukkan pada Gambar 3.14.



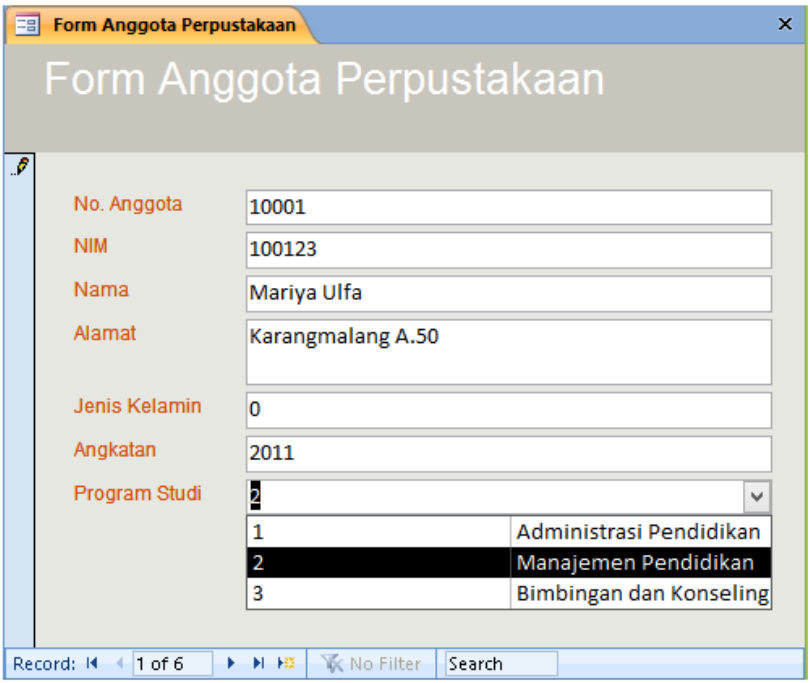
Gambar 3.14: Jendela **Query Builder**

8) Klik tombol pada bagian kiri-atas Access dan klik tombol **Close** (kotak silang pojok kanan atas) jendela **Query Builder**.
9) Klik tombol **Close** (kotak silang pojok kanan atas) Form Anggota Perpustakaan. Jika tidak terjadi kesalahan, maka akan ditampilkan kembali jendela **Property** komponen *combo box* **Kode_Prodi** seperti Gambar 3.15 berikut ini.



Gambar 3.15: Jendela **Property** komponen *text box* **Kode_Prodi**

10) Gambar 3.16 menunjukkan hasil perubahan Form Anggota Perpustakaan. Dapat dilihat bahwa, pada kolom **Kode_Prodi**, selain **Kode_Prodi** juga ditampilkan **Nama_Prodi** sesuai kode, sehingga pengoperasiannya menjadi lebih nyaman.



Gambar 3.16: Form Anggota Perpustakaan Setelah Perubahan

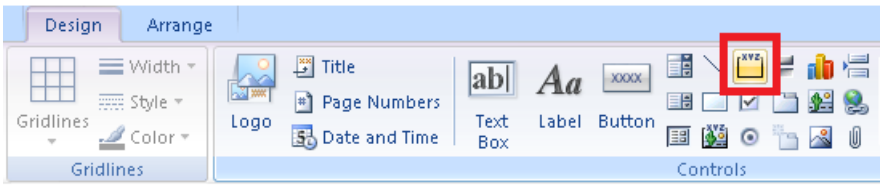
3.2.2 Mengubah Text Box Menjadi Option Group

Langkah selanjutnya adalah mengubah komponen *text box* **Jenis_Kelamin** menjadi komponen *option group*. Langkah awal yang perlu dilakukan adalah kembali ke format *design view* (lihat lagi Gambar 3.8), kemudian ikuti langkah-langkah berikut ini:

1) Hapus komponen *label* dan *text box* **Jenis_Kelamin** sehingga tampilannya menjadi seperti Gambar 3.17.

Gambar 3.17: Form Anggota Perpustakaan Tanpa Kolom **Jenis Kelamin**

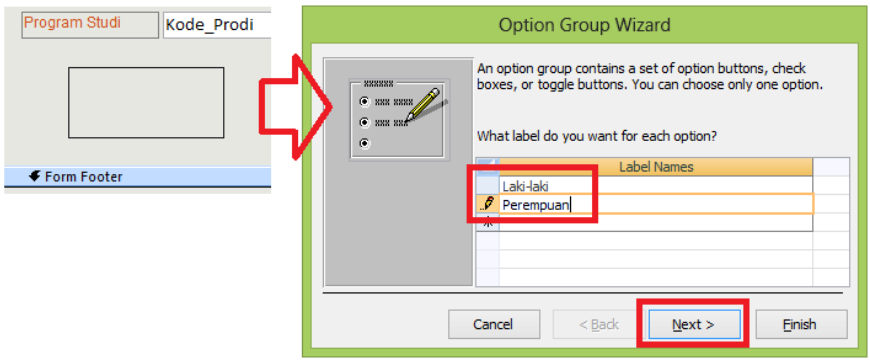
2) Tambahkan komponen *option group* ke dalam form. Caranya, akses tab menu **Design** dan klik komponen *option group*.



Gambar 3.18: Komponen *option group* pada tab menu **Design**

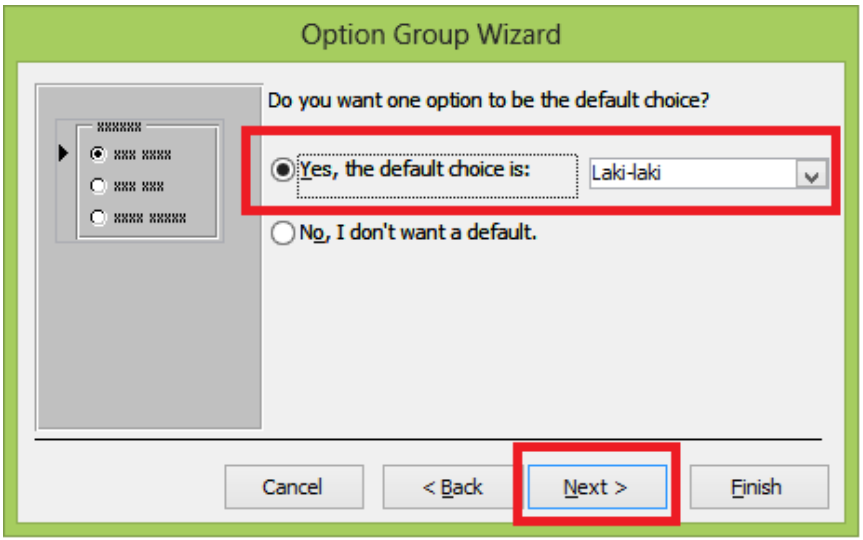
3) Tekan dan tahan (*drag*) komponen *option group* ke bagian bawah form (di bawah komponen *text box* **Kode_Prodi**) sehingga membentuk persegi panjang.
4) Setelah muncul kotak dialog **Option Group Wizard** (OGW), masukkan nama label ke dalam kolom yang tersedia, dalam

hal ini "**Laki-laki**" dan "**Perempuan**". Klik tombol **[Next]** untuk melanjutkan.



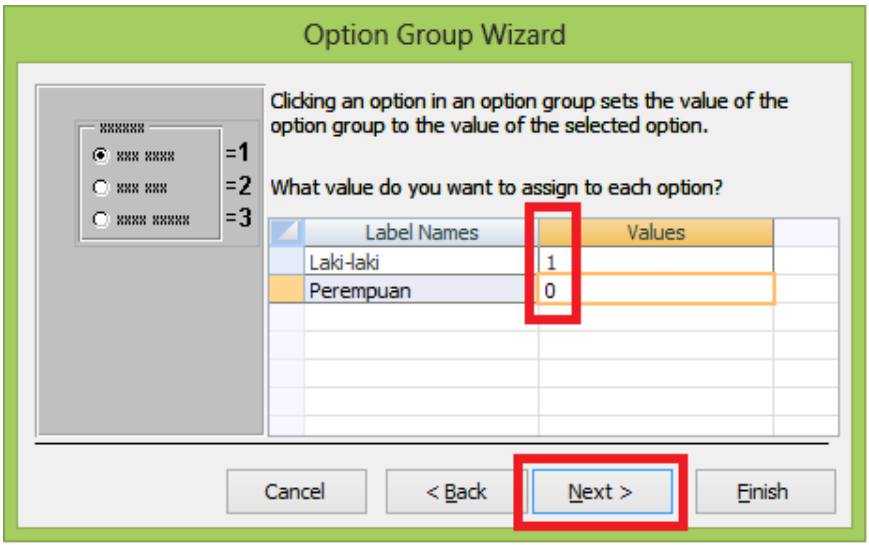
Gambar 3.19: Proses tambah komponen *option group* ke dalam form

5) Setelah muncul kotak dialog **OGW: Default Choice**, pilih opsi "**Yes, the default choice is:**" dan pilih opsi "**Laki-laki**". Klik tombol **[Next]** untuk melanjutkan.



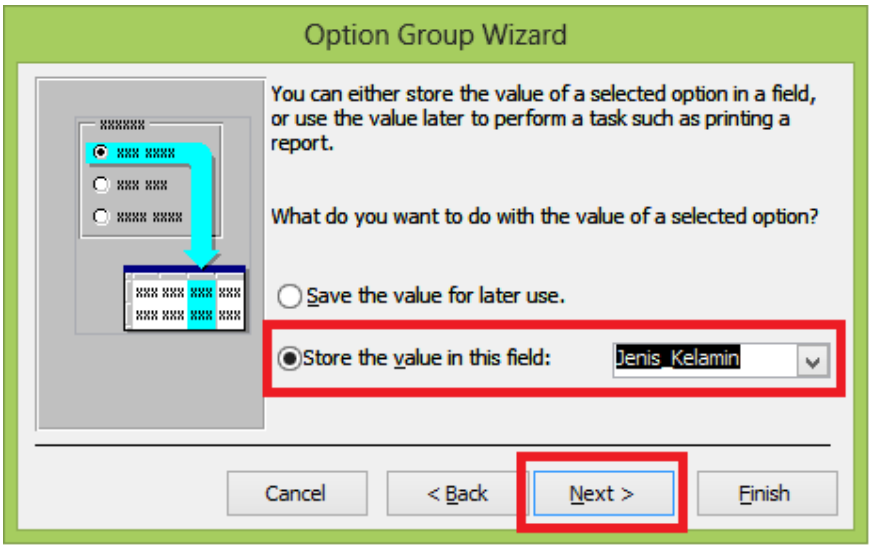
Gambar 3.20: Kotak dialog **OGW: Default Choice**

6) Setelah muncul kotak dialog **OGW: Value of Option**, isi **1** (satu) untuk opsi "**Laki-laki**" dan **0** (nol) untuk "**Perempuan**". Klik tombol **[Next]** untuk melanjutkan.



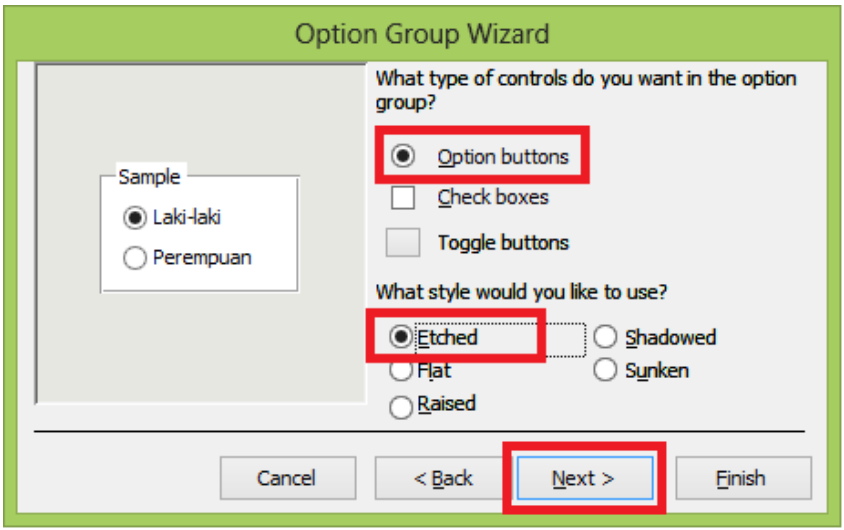
Gambar 3.21: Kotak dialog **OGW: Value of Option**

7) Setelah muncul kotak dialog **OGW: Save or Store**, pilih opsi "**Store the value in this field:**" dan pilih opsi kolom "**Jenis_Kelamin**". Klik tombol **[Next]** untuk melanjutkan.



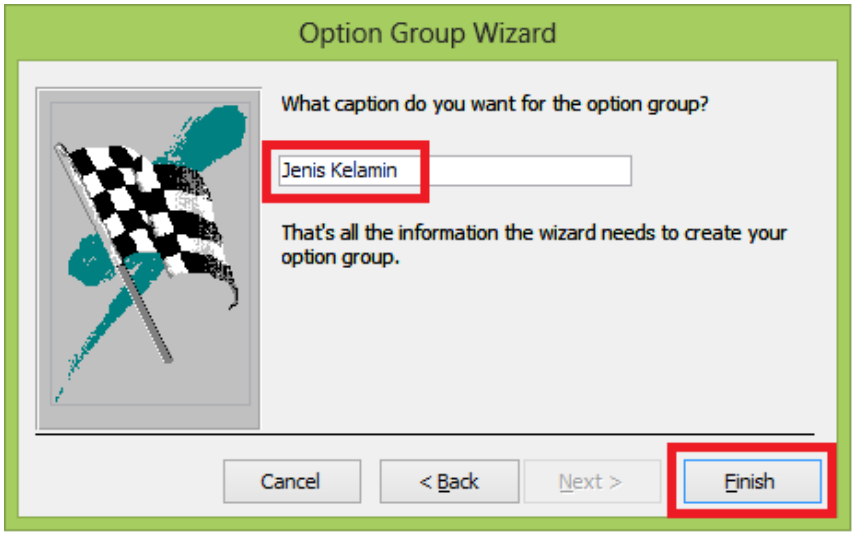
Gambar 3.22: Kotak dialog **OGW: Save or Store**

8) Setelah muncul kotak dialog **OGW: Type and Style of Control**, pilih opsi "**Option buttons**" untuk tipe, dan pilih opsi "**Etched**" untuk *style*. Klik tombol **[Next]** untuk melanjutkan.



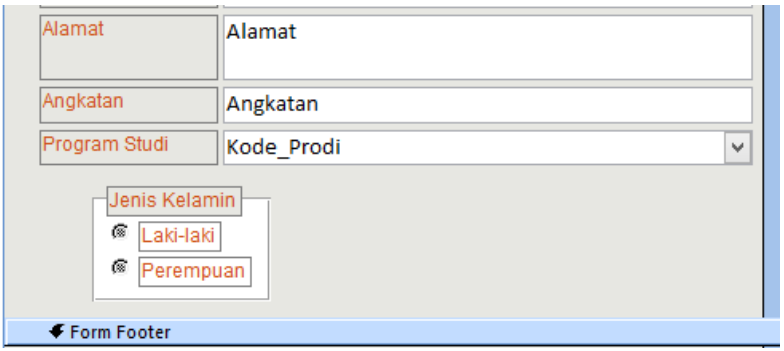
Gambar 3.23: Kotak dialog **OGW: Save or Style of Control**

9) Setelah muncul kotak dialog **OGW: Caption of Option Group**, ketik "**Jenis Kelamin**. Klik tombol **[Finish]** untuk mengakhiri.



Gambar 3.24: Kotak dialog **OGW: Caption of Option Group**

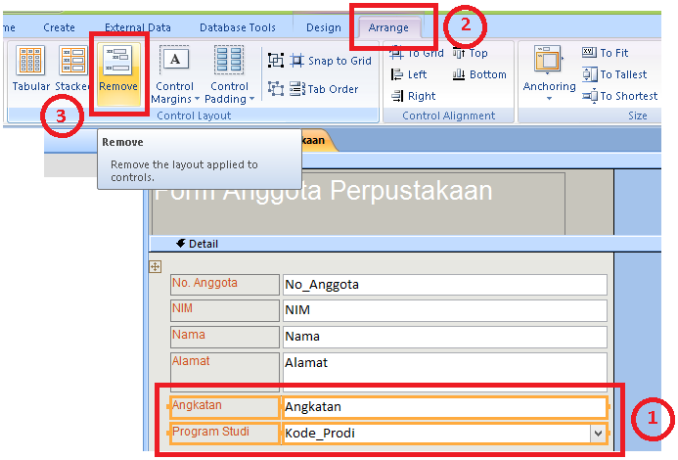
10) Gambar 3.25 menunjukkan komponen *option group* dalam **Form Anggota Perpustakaan**.



Gambar 3.25: Komponen *option group* dalam form

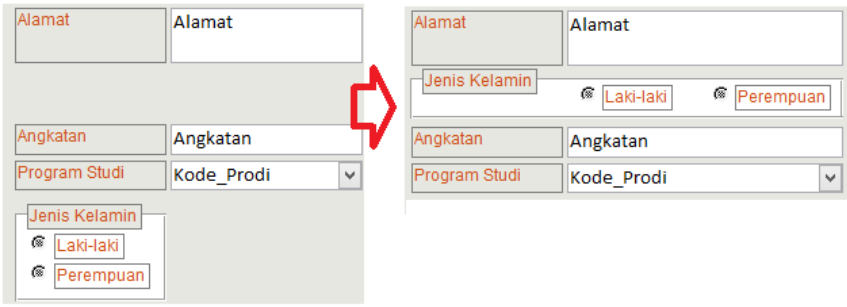
11) Langkah selanjutnya adalah menaruh komponen *option group* di bawah komponen *text box* **Alamat**. Caranya, pilih

komponen *text box* **Angkatan** dan **Kode_Prodi**, akses tab menu **Arrange** dan klik ikon tombol **Remove**. Tujuannya agar kedua komponen *text box* tersebut bisa digeser ke bawah. Perhatikan ilustrasinya pada Gambar 3.26.



Gambar 3.26: Proses menggeser *text box* **Angkatan** dan **Kode_Prodi**

12) Langkah selanjutnya adalah menggeser komponen *option group* ke atas (di bawah *text box* **Alamat**). Perhatikan ilustrasinya pada Gambar 3.27.



Gambar 3.27: Proses menggeser *option group* ke atas

13) Agar lebih serasi, hapus teks "**Jenis Kelamin**" pada komponen *option group*, kemudian tambahkan komponen label dan beri *caption* "**Jenis Kelamin**.

Gambar 3.28: Proses menambahkan komponen *label*

14) Terakhir, kecilkan dan atur komponen *option group* sehingga menjadi rapi seperti Gambar 3.29.

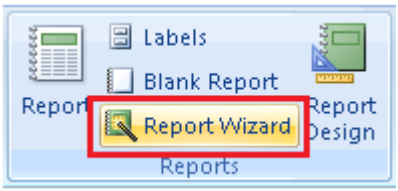
Gambar 3.29: Tampilan akhir **Form Anggota Perpustakaan**

Dengan cara yang sama, Anda dapat membuat form untuk tabel lainnya (selain **tAnggota**). Selamat mencoba!

3.3 Membuat Report

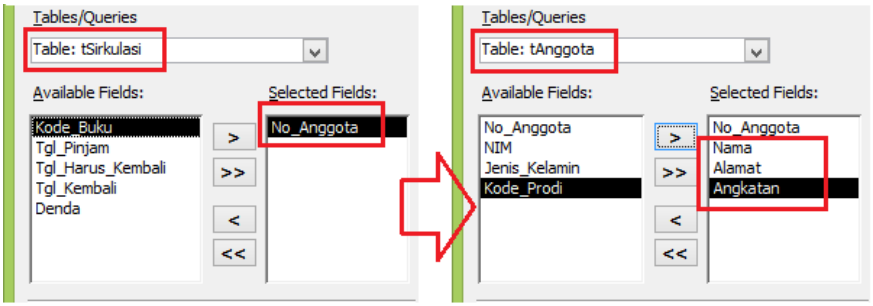
Pada bagian ini akan dibahas cara membuat laporan dengan *wizard*. Misalkan laporan yang diinginkan adalah **Laporan**

Sirkulasi Perpustakaan yang ditampilkan berdasarkan data **Anggota**. Berikut adalah langkah-langkahnya:



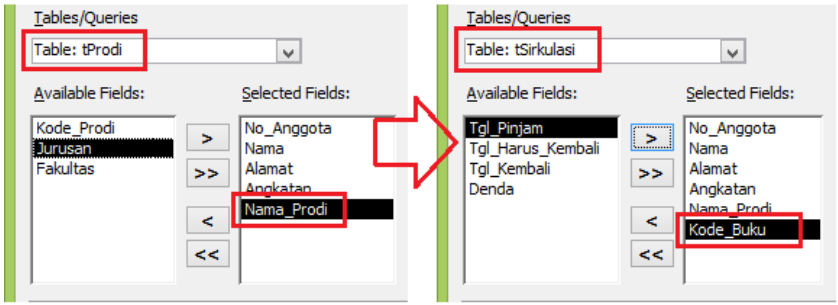
Gambar 3.30: Ikon tombol **Report Wizard**

1) Dari tab menu **Create**, klik ikon tombol **Report Wizard** pada Gambar 3.30, maka akan ditampilkan kotak dialog pemilihan kolom-kolom dari satu atau banyak tabel seperti yang ditunjukkan pada Gambar 3.31. Karena laporan ini melibatkan seluruh tabel yang ada, maka perlu dengan urut dan teliti dalam memilih kolom-kolom yang akan ditampilkan sebagai laporan. Pertama, pilih tabel **tSirkulasi (No_Anggota)** dan **tAnggota (Nama, Alamat, Angkatan)**.



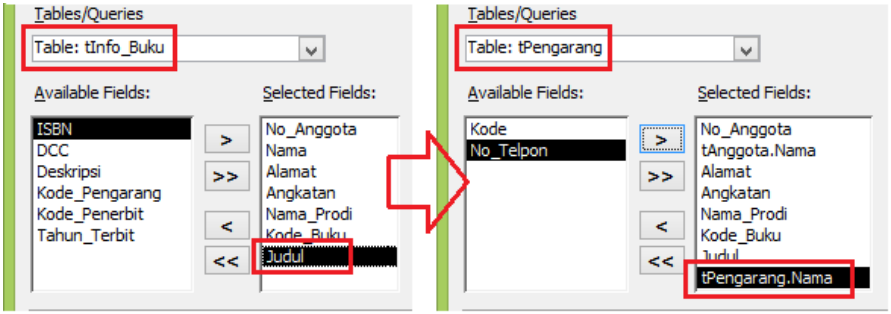
Gambar 3.31: Pemilihan kolom pada tabel **tSirkulasi** dan **tAnggota**

2) Berikutnya, pilih tabel **tProdi (Nama_Prodi)** dan **tSirkulasi (Kode_Buku)**.



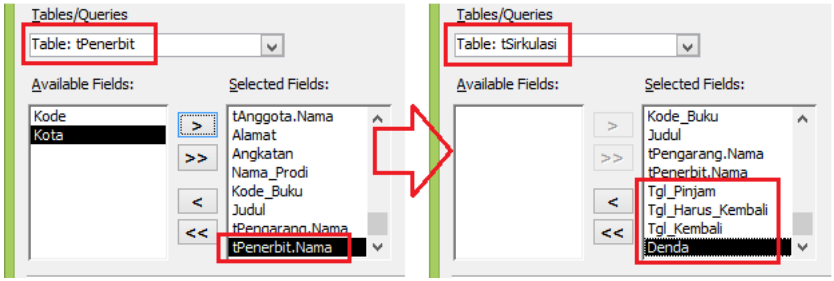
Gambar 3.32: Pemilihan kolom pada tabel **tProdi** dan **tSirkulasi**

3) Selanjutnya, pilih tabel **tInfo_Buku (Judul)** dan **tPengarang (Nama)**.



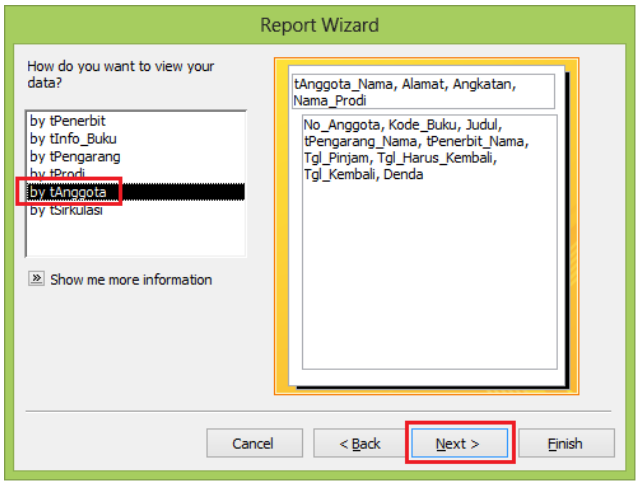
Gambar 3.33: Pemilihan kolom pada tabel **tInfo_Buku** dan **tPengarang**

4) Terakhir, pilih tabel **tPenerbit (Nama)** dan **tSirkulasi (Tgl_Pinjam, Tgl_Harus_Kembali, Tgl_Kembali, Denda)**.



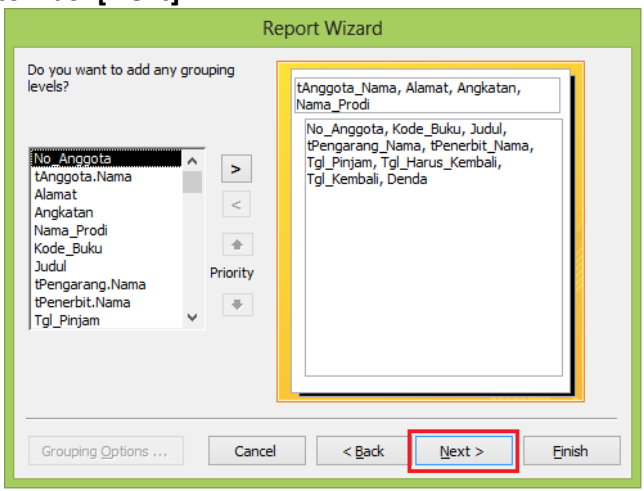
Gambar 3.34: Pemilihan kolom pada tabel **tPenerbit** dan **tSirkulasi**

5) Klik tombol **[Next]**, maka akan ditampilkan jendela pengegrupan berdasarkan **tabel** tertentu, dalam hal ini kita pilih **by tAnggota** (lihat Gambar 3.35).



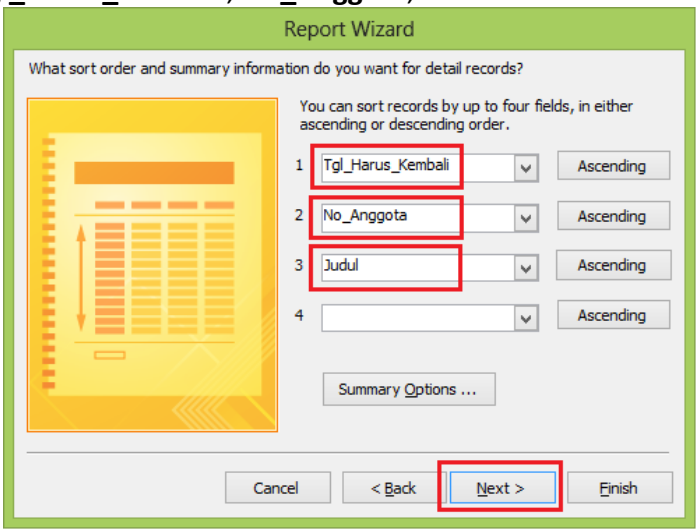
Gambar 3.35: Jendela **Report Wizard: Group By the Table**

6) Klik tombol **Next**, maka akan ditampilkan jendela tambahan pengegrupan berdasarkan **kolom** tertentu, biarkan saja dan klik tombol **[Next]**.



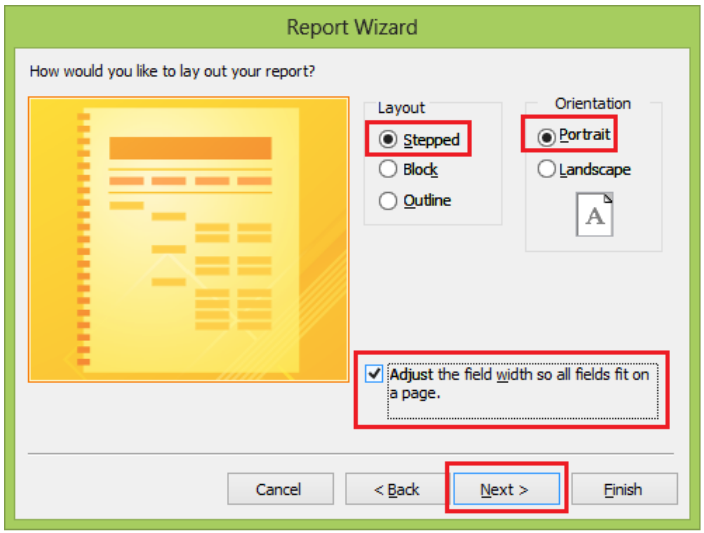
Gambar 3.36: Jendela **Report Wizard: Group By the Field**

7) Klik tombol **[Next]**, maka akan ditampilkan jendela pengurutan berdasarkan kolom tertentu, misalkan **Tgl_Harus_Kembali**, **No_Anggota**, dan **Judul**.



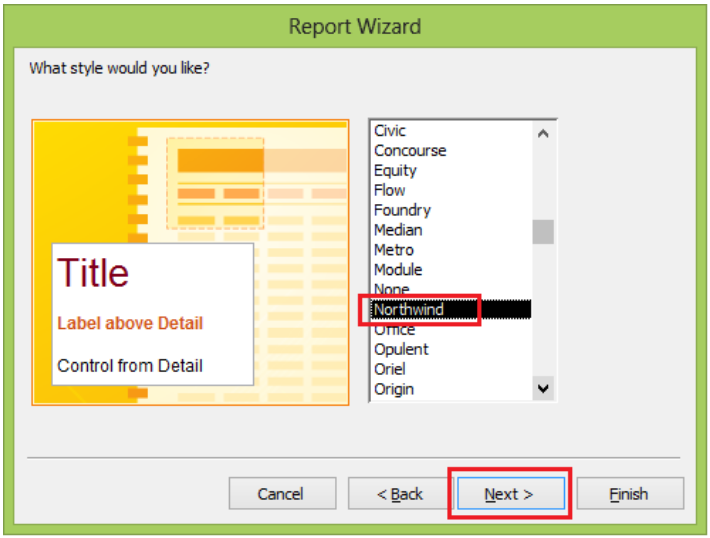
Gambar 3.37: Jendela **Report Wizard: Order By the Field**

8) Klik tombol **Next**, maka akan ditampilkan jendela *layout* laporan, pilih **Stepped** dan **Portrait**, serta beri tanda centang opsi **Adjust the field width so all fields fit on a page**.



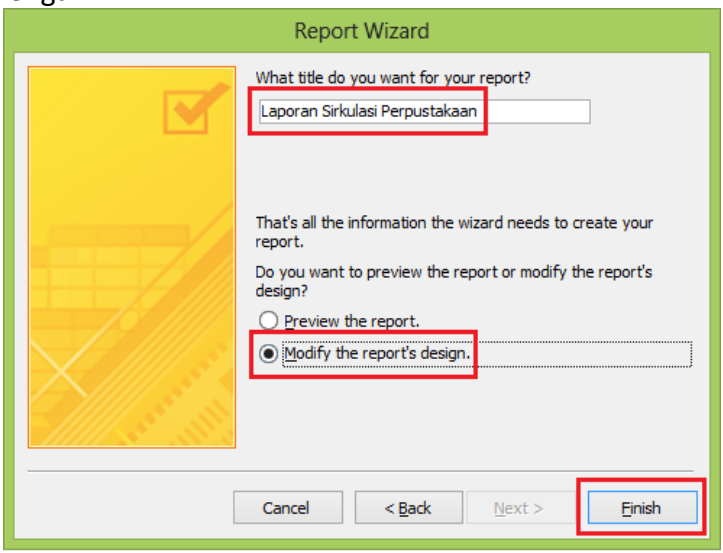
Gambar 3.38: Jendela **Report Wizard: Layout and Orientation**

9) Kemudian klik tombol **[Next]**, maka akan ditampilkan jendela style laporan, misalkan kita pilih **Northwind**.



Gambar 3.39: Jendela **Report Wizard: Style**

10) Klik tombol **[Next]**, maka akan ditampilkan jendela untuk menentukan judul laporan, ketik "**Laporan Sirkulasi Perpustakaan**". Karena masih akan diedit/rapikan, pilih opsi **Modify the report's design** dan klik tombol **[Finish]** untuk mengakhiri.



Gambar 3.40: Jendela **Report Wizard: Title**

11) Gambar 3.41 berikut ini merupakan halaman desain **Laporan Sirkulasi Perpustakaan**.

Gambar 3.41: Halaman desain **Laporan Sirkulasi Perpustakaan**

12) Lakukan modifikasi sesuai selera sehingga **Laporan Sirkulasi Perpustakaan** semakin rapi dan layak disajikan, misalkan seperti Gambar 3.42.

Gambar 3.42: Halaman desain akhir **Laporan Sirkulasi Perpustakaan**

13) Gambar 3.43 berikut ini merupakan halaman **Laporan Sirkulasi Perpustakaan**.

Laporan Sirkulasi Perpustakaan

No. Anggot	Nama	Alamat	Angkatan	Program Studi
10001	Mariya Ulfa	Karangmalang A.50	2011	Manajemen Pendidikan

Judul	Kode	Pengarang	Penerbit
Tanggal Pinjam	Tgl. Harus Kembali	Tanggal Kembali	Denda		
Supervisi Pendidikan	100002	Lantip Diat Prasojo	Gava Media
17/10/2014	20/10/2014	20/10/2014	0,00		
Teknologi Informasi Pendidikan	100001	Lantip Diat Prasojo	Gava Media
02/10/2014	05/10/2014	04/10/2014	0,00		

10006	Supirso	Blimbingsari F.31	2011	Bimbingan dan Konseling

Judul	Kode	Pengarang	Penerbit
Tanggal Pinjam	Tgl. Harus Kembali	Tanggal Kembali	Denda		
Pemrograman Java SE/ME/EE	100007	Riyanto	Gava Media
20/10/2014	23/10/2014	23/10/2014	0,00		
Pemrograman PHP-MySQL	100008	Riyanto	Gava Media
19/10/2014	22/10/2014	23/10/2014	2.000,00		
Supervisi Pendidikan	100002	Lantip Diat Prasojo	Gava Media
20/10/2014	23/10/2014	23/10/2014	0,00		

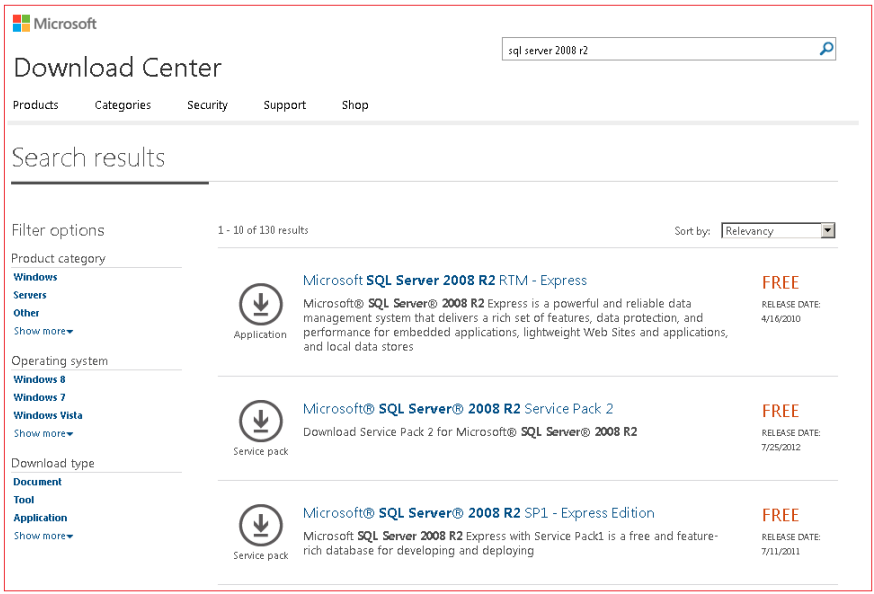
24 Oktober 2014

Gambar 3.43: Potongan halaman **Laporan Sirkulasi Perpustakaan**

Demikian materi pembuatan *report* dalam Access, untuk halaman laporan lainnya, Anda dapat bereksperimen sendiri.

4 Instalasi Microsoft SQL Server dan Konfigurasinya

Pada bab ini akan dibahas langkah-langkah instalasi Microsoft SQL Server (selanjutnya disebut **SQL Server**) dan konfigurasinya, termasuk konfigurasi *firewall* yang berhubungan dengan pengaksesan *port*. Di buku ini akan digunakan **Microsoft SQL Server 2008 R2** (*Express Edition*) yang dapat diunduh secara gratis melalui situs resmi **Microsoft Corp.**, tepatnya di tautan: http://www.microsoft.com/en-us/download/search.aspx?q=sql+server+2008+r2.



Gambar 4.1: Beberapa versi Ms. SQL Server 2008

4.1 Kebutuhan Sistem

Sebelum melakukan instalasi, perlu diketahui spesifikasi sistemoperasi dan perangkat keras (*hardware*) minimal yang diperlukan oleh SQL Server sehingga proses instalasi bisa berhasil.

Untuk sistem operasi, SQL Server 2008 Express Edition mendukung beberapa versi Windows, yaitu: Windows 7, Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, dan Windows Vista.

SQL Server R2 ini merupakan SQL Server versi ringkas, sehingga memiliki berbagai keterbatasan jika dibandingkan dengan versi lengkapnya. Salah satu keterbatasan yang penulis temukan saat menggunakan SQL Server versi ini adalah hanya bisa mengeksekusi paket data kurang dari **10 GB** (sepuluh *giga byte*). Namun demikian, SQL Server Express Edition lebih dari memadai untuk digunakan sebagai sistem basis data untuk aplikasi yang cukup kompleks namun tidak begitu memerlukan tingkat keamanan yang tinggi.

Karena versi ringkas, tentunya kebutuhan perangkat keras tidak sama dengan versi lengkapnya. Berikut adalah spesifikasi perangkat keras minimal yang diperlukan untuk bisa melakukan instalasi SQL Server 2008 R2.

- Prosesor : Pentium III atau prosesor setara lainnya dengan kecepatan 1 GHz atau lebih tinggi.
- Memori: Minimal 500 MB untuk SQL Server Express dengan *Tools* dan *Advanced Services*, dan 4 GB untuk SQL Server dengan *Reporting Services*.
- Harddisk : 2.2 GB, belum termasuk *space* untuk sistem operasi dan *database* aplikasi yang akan Anda kembangkan menggunakan SQL Server.

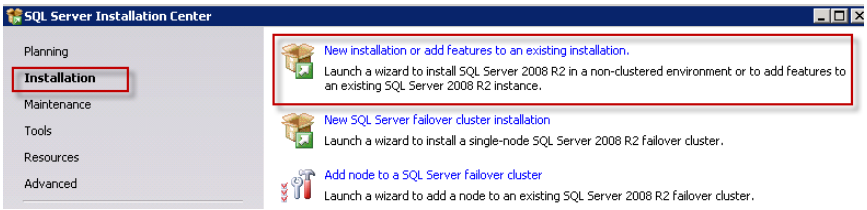
4.2 Instalasi SQL Server 2008 R2

SQL Server 2008 ini berjalan di atas *framework* **.NET 3.5** (baca: dot NET versi 3.5),. Oleh karena itu, sebelum melakukan instalasi SQL Server, Anda harus menginstal *framework* tersebut. File *installer framework* .NET dapat diunduh dengan bebas di situs resminya Microsoft. Selanjutnya, berikut adalah langkah-langkah instalasi SQL Server 2008.

1) Klik ganda (*double clicked*) file *installer* SQL Server, jangan lupa sesuaikan dengan Windows Anda. Jika Windows OS Anda 32bit, gunakan file **...x86.exe**, dan **...x64.exe** jika 64bit.

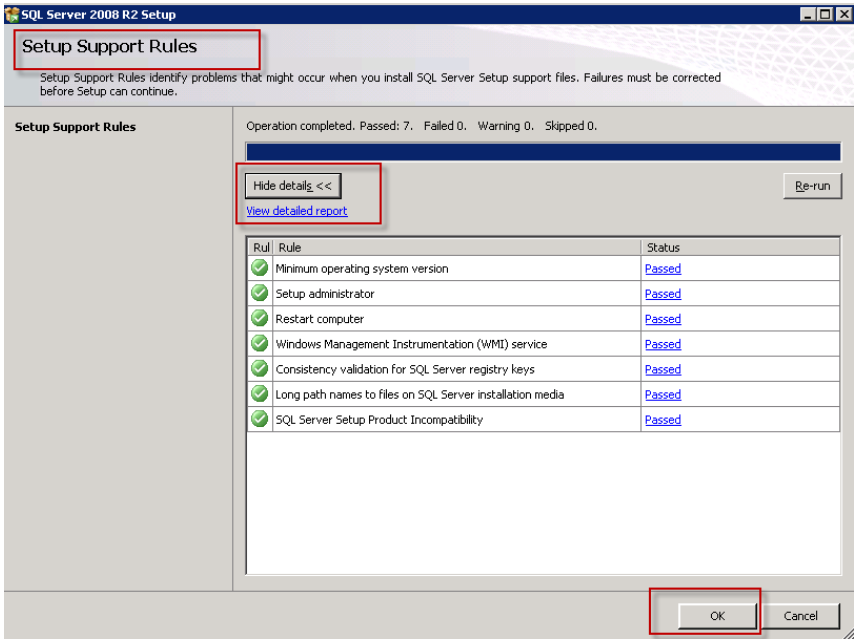
Gambar 4.2: File *installer* SQL Server untin Windows 32Bit dan 64Bit

2) Maka akan muncul jendela *progress* ekstraksi file *installer*. Tunggu saja sampai muncul jendela **SQL Server Installation Center**.
3) Klik menu **Installation** pada sebelah kiri jendela untuk melihat beberapa opsi instalasi, kemudian klik opsi **New installation or add features to an existing installation**.



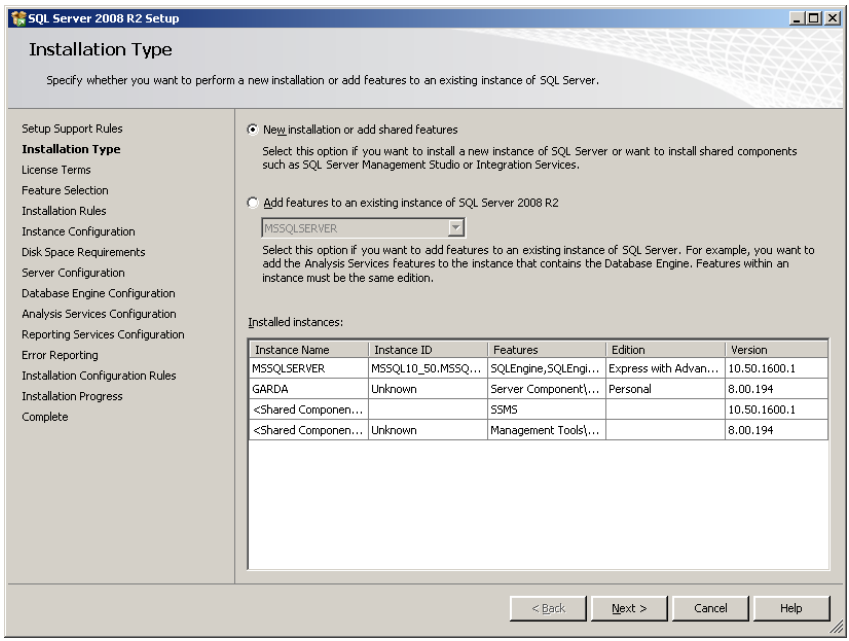
Gambar 4.3: Jendela **SQL Server Intallation Center**

4) Tunggu lagi beberapa saat sampai muncul jendela **Setup Support Rules**, klik tombol **[OK]**. Jika tombol **[Show details]** diklik, maka akan muncul seperti Gambar 4.4 berikut.



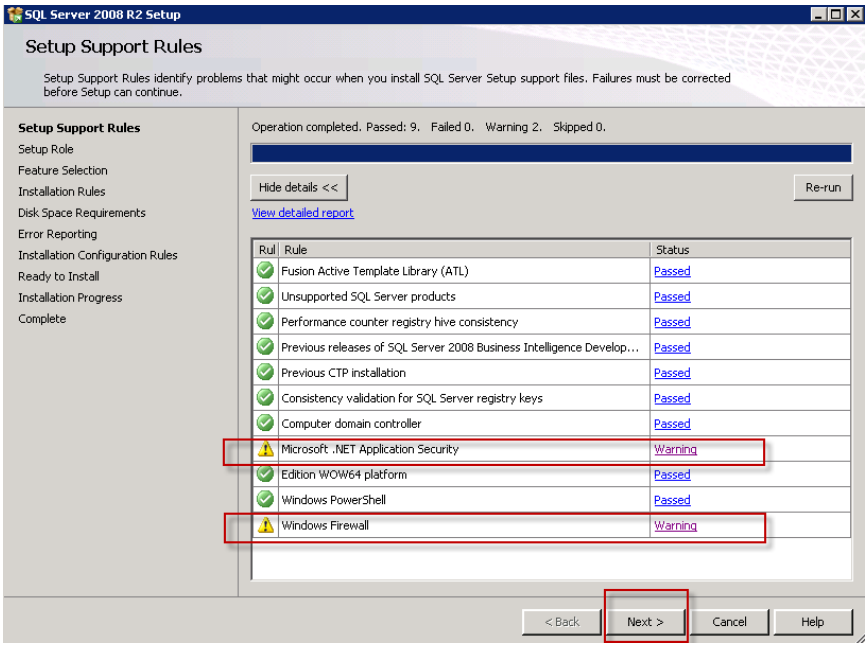
Gambar 4.4: Jendela **Setup Support Rules**

5) Klik menu **Installation Type** yang ada di sebelah kiri jendela, pilih opsi **New installation or add shared features** dan klik tombol **[Next]**. **INGAT!** Selama proses instalasi, jendela **SQL Server Intallation Center** jangan ditutup.



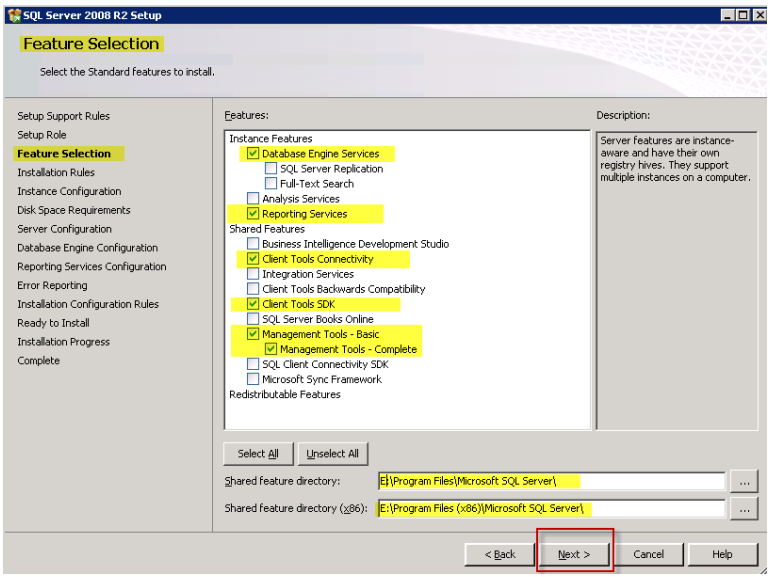
Gambar 4.5: Jendela **Installtion Type**

6) Jendela berikutnya adalah **License terms**. Beri tanda centang opsi **I accept the license terms** dan klik tombol **[Next]**.
7) Pada jendela **Setup Support Rules** mungkin Anda akan menemukan sepasang peringatan (*warning*) seperti yang ditunjukkan Gambar 4.6.
   - Peringatan aplikasi **.NET** menunjukkan bahwa komputer Anda tidak terhubung ke internet. Ini bukan masalah, jadi bisa diabaikan.
   - Selanjutnya, peringatan **Windows Firewall** menunjukkan bahwa perlu dilakukan pengaturan *firewall* sebelum dilakukan instalasi **SQL Server**. Peringatan ini juga bisa diabaikan.



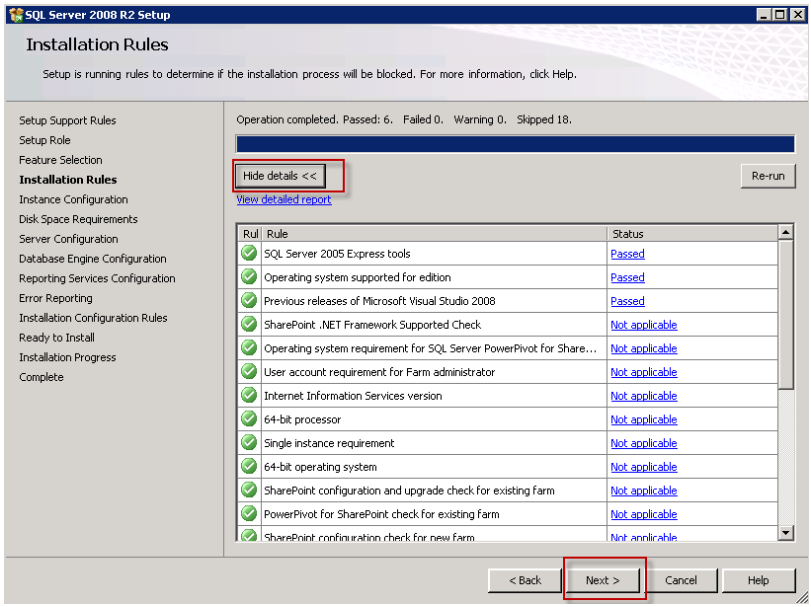
Gambar 4.6: Sepasang peringatan pada jendela **Setup Support Rules**

8) Klik saja tombol **[Next]** untuk menuju ke jendela **Feature Selection**.
9) Pada jendela **Feature Selection**, Anda bisa memilih semua fitur atau hanya memberi tanda centang fitur apa saja yang ingin diinstal. Tentunya, semakin banyak fitur yang diinstal akan membutuhkan lebih banyak ruang penyimpanan. Namun demikian, fitur **SQL Server Management Tool** harus dipilih.
10) Tentukan juga alamat direktori (*path*) fitur yang di-*share* melalui kolom **Shared feature directory**. Anda bisa membiarkan *default path*, atau menentukan *drive* yang diinginkan, pada contoh ini digunakan *drive* "**E**".



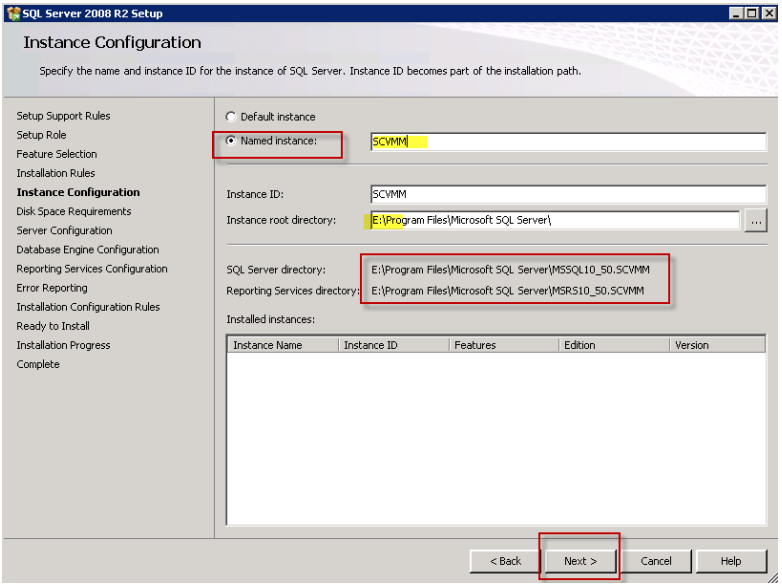
Gambar 4.7: Jendela **Feature Selection**

11) Klik tombol **[Next]**, maka akan ditampilkan jendela **Installation Rules**.



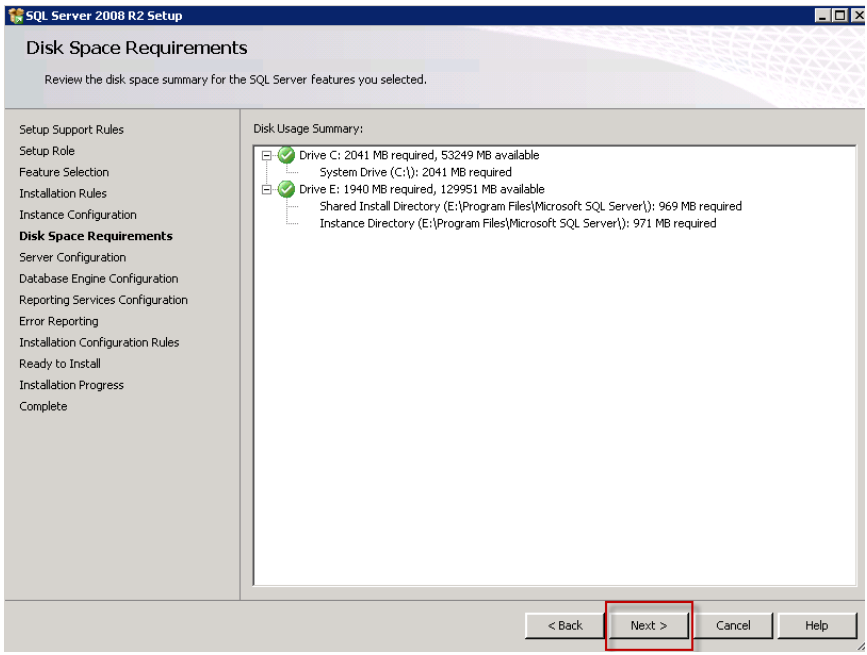
Gambar 4.8: Jendela **Installation Rules**

12) Pada jendela ini, Anda bisa mengklik tombol **[Show details]** untuk melihat berbagai status *rule* yang diinstal atau langsung mengklik tombol **[Next]** untuk melanjutkan.
13) Jendela berikutnya yang muncul adalah **Instance Configuration**. Pada sesi ini, Anda diminta untuk menentukan nama *instance* serta *root directory*-nya. Anda dapat memilih opsi **Default instance** jika Anda ingin tidak banyak melakukan konfigurai, atau menentukan sesuai keinginan, seperti yang dicontohkan pada buku ini. Jika Anda menentukan nama *instance* sendiri, harus mengikuti aturan bahwa nama *instance* **TIDAK BOLEH** lebih dari 15 (lima belas) karakter.
14) Di bagian tengah yang ditandai dengan tanda kotak menunjukkan direktori **SQL Server** dan **Reporting Service** hasil penentuan nama *instance* dan *root directory*-nya.



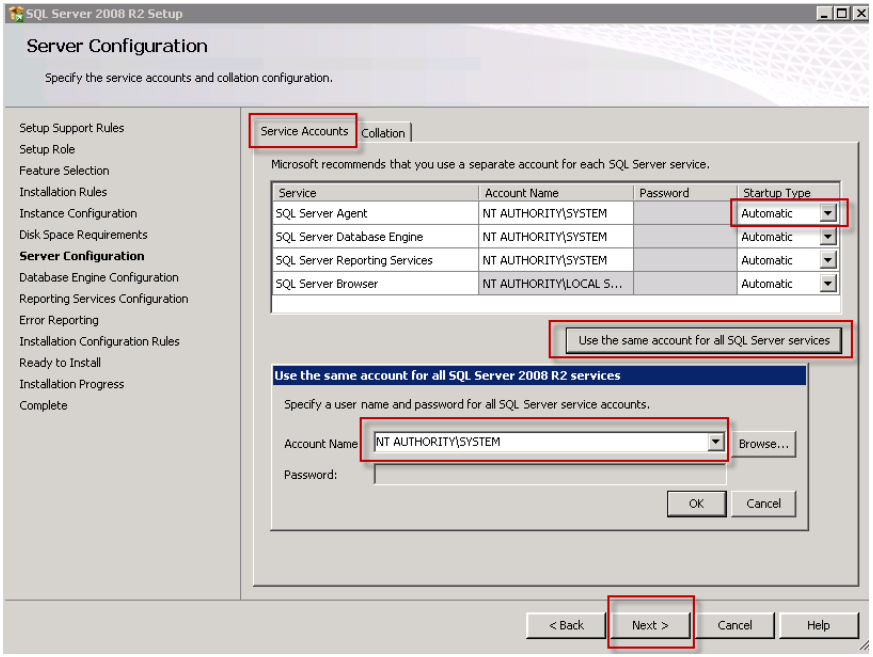
Gambar 4.9: Jendela **Instance Configuration**

15) Klik tombol **[Next]**, maka akan ditampilkan jendela **Disk Space Requirements**.



Gambar 4.10: Jendela **Disk Space Requirements**

16) Pada jendela ini tidak ada konfigurasi apapun, langsung saja klik tombol **[Next]** untuk menuju ke jendela **Server Configuration**.



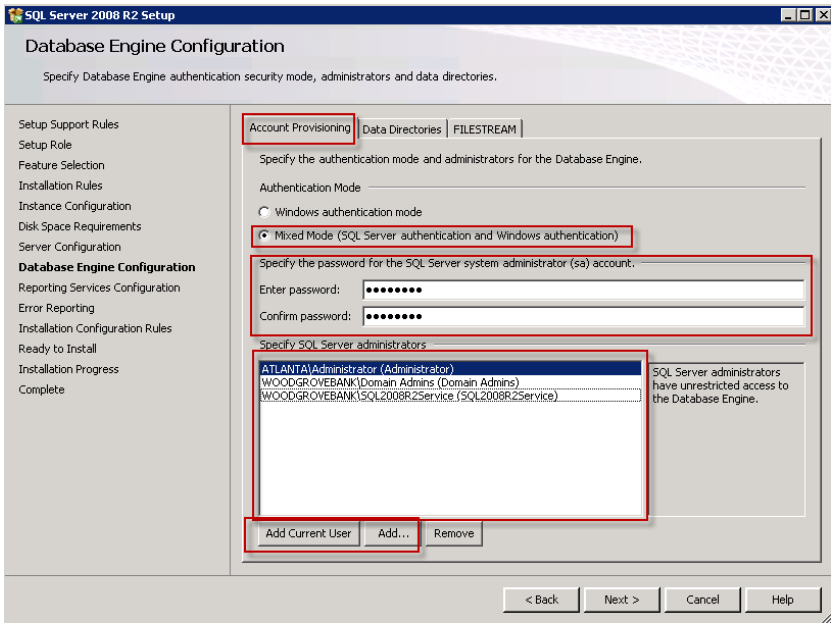
Gambar 4.11: Jendela **Server Configuration**

17) Beberapa hal yang perlu dilakukan konfigurasi adalah sebagai berikut.
    - Jika *remote* SQL Server jalan di bawah akun **Network Service** atau akun domain, maka Anda harus membuat sebuah **Service Principal Name** (SPN) untuk layanan SQL sebagaimana dideskripsikan di dalam artikel **Microsoft Knowledge Base**, tepatnya di tautan http://support.microsoft.com/default.aspx?scid=kb;en-us;811889.
    - Di tab **Service Account**, ubah **Startup Type** untuk layanan **SQL Server Agent** menjadi "**Automatic**". *Agent* ini digunakan untuk proses *backup*.
    - Jika diinginkan menggunakan akun yang sama untuk seluruh layanan, klik tombol **[Use the same account for all SQL Server services]**. Setelah muncul kotak dialog **Use the same account for all SQL Server 2008 R2 services**, pada kolom

Account Name pilih "**NT/AUTHORITY \SYSTEM**", kemudian klik tombol **[OK]**.

- Pada tab **Collation**, biarkan apa adanya (*default*).

18) Klik tombol **[Next]** untuk menuju ke jendela **Database Engine Configuration**.

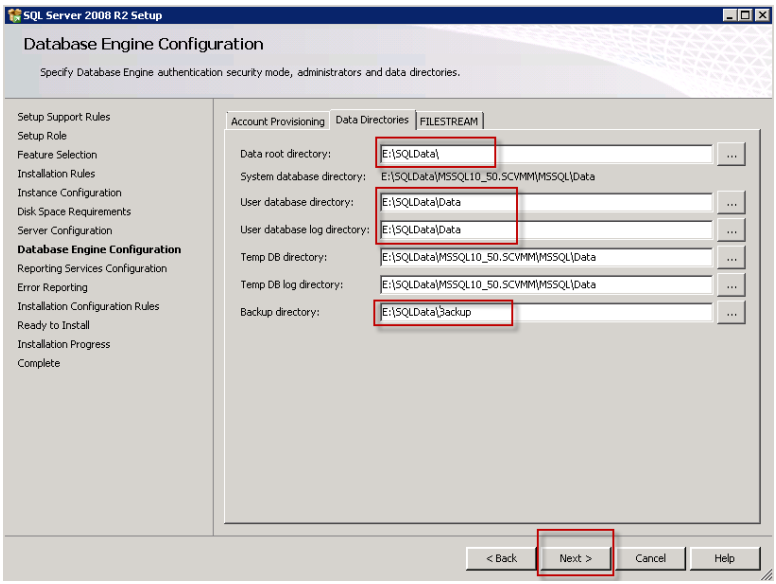


Gambar 4.12: Tab **Account Provisioning**
pada Jendela **Databse Engine Configuration**

19) Beberapa hal yang perlu dilakukan pada jendela ini adalah sebagai berikut.

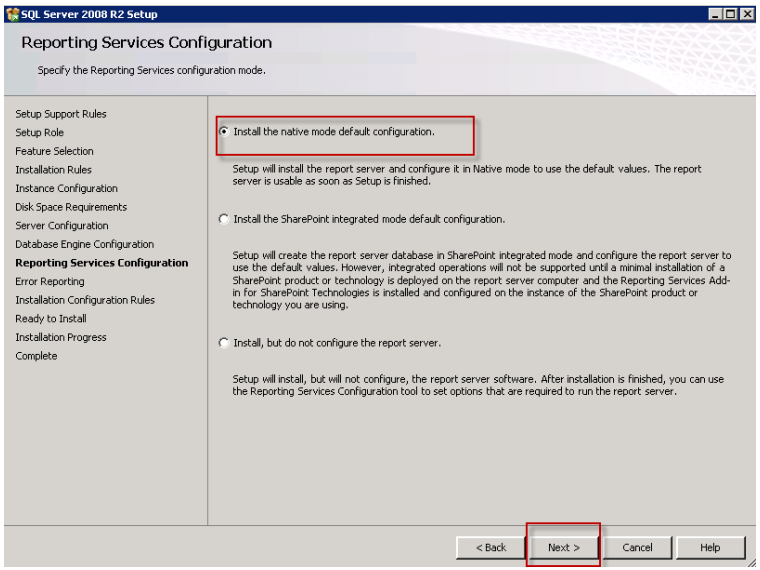
- Pada tab **Account Provisioning**, Anda bisa memilih opsi **Windows authentication mode** tanpa menentukan *password*, atau memilih opsi **Mixed Mode (SQL Server authentication and Windows authemtication)** dan masukkan *password* untuk akun "**sa**". Akun "**sa**" merupakan akun administrator internal SQL Server.
- Anda juga bisa menambahkan daftar *user* yang diinginkan ke dalam kolom **Specify SQL Server administrators** dengan mengklik tombol **[Add Current User]**.

- Pada tab **Data Directories**, Anda dapat mengubah lokasi penyimpanan data SQL Server. Direkomendasikan untuk **TIDAK** menyimpan data SQL Server di *drive* "**C**". Pada contoh in, folder "**SQLData**" yang sebelumnya telah dibuat di *drive* "**E**" digunakan sebagai lokasi menyimpan **Data root directory**. Folder "**SQLData\Data**" untuk **User database directory** dan **User database log directory**, serta folder "**SQLData\Backup**" untuk **Backup directory**.
- Khusus untuk **Temp DB directory** dan **Temp DB log directory** dibiarkan *default*.



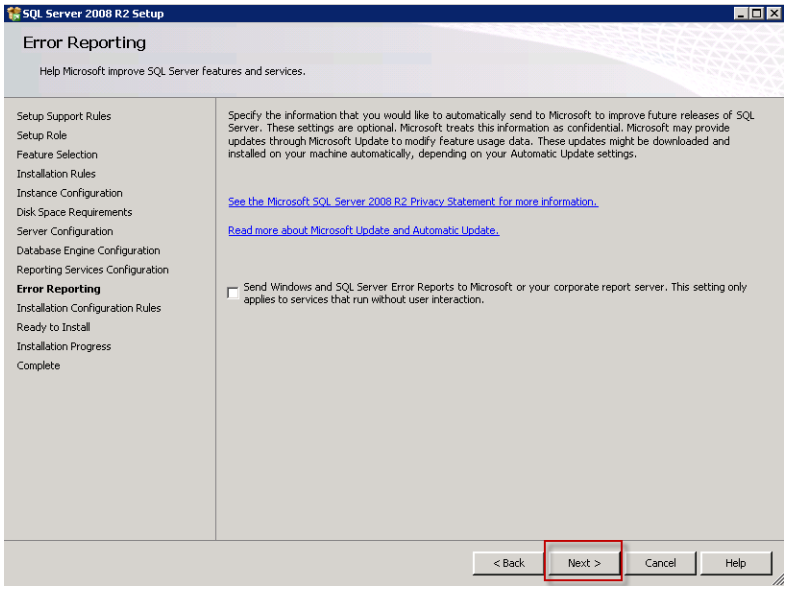
Gambar 4.13: Tab **Data Directories**
pada Jendela **Databse Engine Configuration**

20) Pada tab **FILESTREAM** tidak perlu dilakukan konfigurasi dan langsung klik tombol **[Next]** untuk menuju ke jendela **Reporting Service Configuration**.



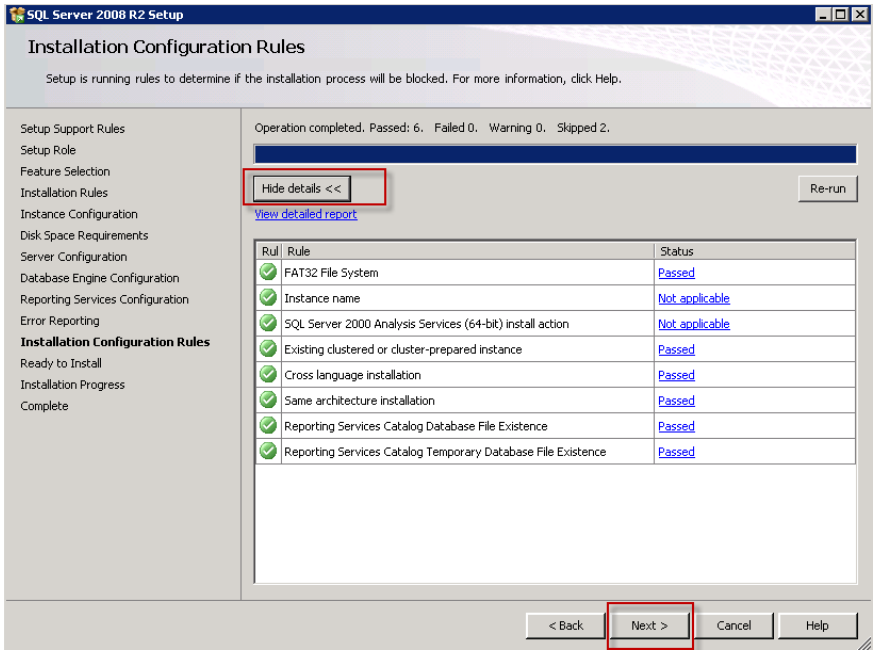
Gambar 4.14: Jendela **Reporting Service Configuration**

21) Pilih opsi **Install the native mode default configuration** dan klik tombol **[Next]** untuk menuju ke jendela **Error Reporting**.



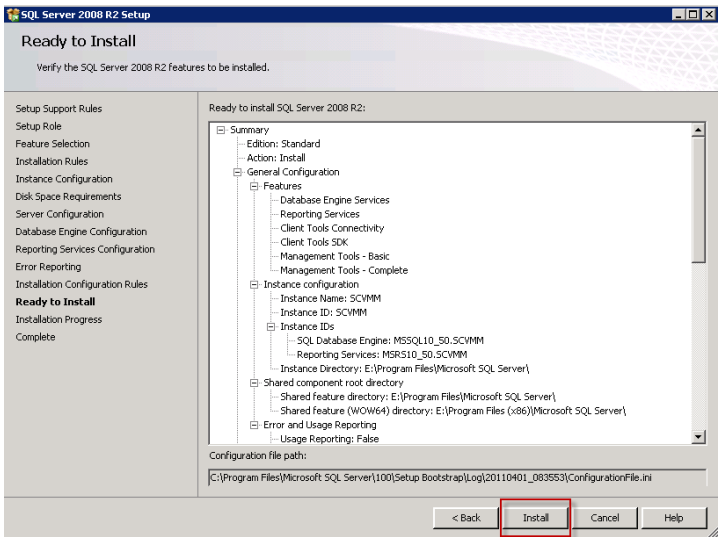
Gambar 4.15: Jendela **Error Reporting**

22) Jika tidak ditemukan kesalahan, langsung klik tombol **[Next]** untuk menuju ke jendela **Installation Configuration Rules**.



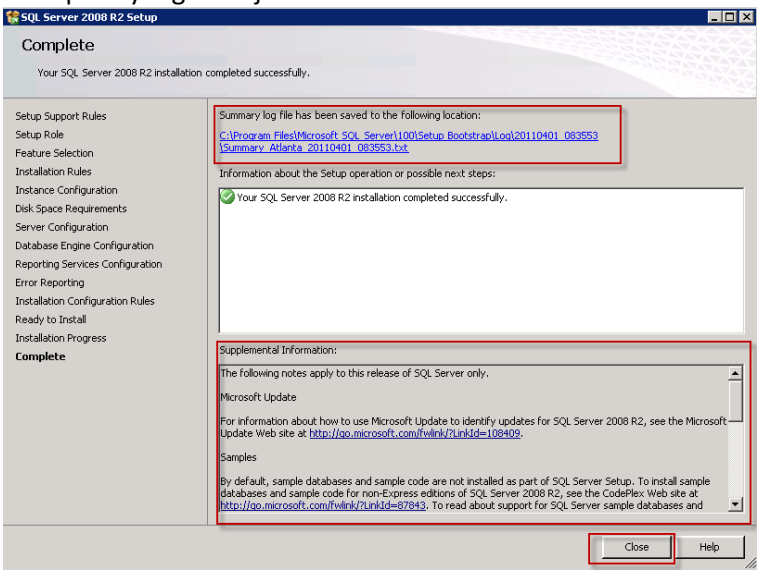
Gambar 4.16: Jendela **Installation Configuration Rules**

23) Klik tombol **[Show details]** untuk melihat status instalasi *rule* dan klik tombol **[Next]** sampai muncul jendela **Ready to Install**.



Gambar 4.17: Jendela **Ready to Install**

24) Klik tombol **[Install]** untuk melanjutkan proses instalasi. Jika berhasil, maka akan muncul jendela **Complete** yang lebih kurang seperti yang ditunjukkan Gambar 4.18.



Gambar 4.18: Jendela **Complete**

25) Proses instalasi **SQL Server 2008 R2** selesai.

5 Administrasi Database Microsoft SQL Server

Mengingat SQL Server tergolong RDBMS (*relational database management system*) yang cukup besar dan kompleks, membahas fitur-fiturnya secara lengkap tidak akan cukup dituangkan di dalam sebuah buku, apalagi hanya satu bab. Oleh karena itu, pembahasan pada bab ini hanya sebatas tentang langkah-langkah membuat *database* dan komponennya (*table*, *view*, *stored procedure*, dan *trigger*), membuat berbagai perintah *query* yang sering digunakan, dan proses administrasi lain dianggap perlu dilakukan oleh seorang *database enginer*. Hal ini dimaksudkan untuk menjembatani pembaca pemula yang mungkin mengalami kesulitan dalam melakukan hal-hal dasar tersebut atau yang kurang terbiasa dengan *query* di SQL Server. Bagi pembaca tingkat mahir bisa mengabaikan bab ini dan langsung fokus ke bab selanjutnya yang diinginkan.

Sebagai bahan latihan, pada bab ini akan dibahas rancangan *database* untuk penjualan sederhana. *Database* ini akan diberi nama "**db_commerce**" dan di dalamnya akan dibuat enam buah tabel, yaitu: "**dbo.products**", "**dbo.orderdetail**", "**dbo.orders**", "**dbo.customer**", "**dbo.supply**", dan "**dbo.supplier**".

5.1 Membuat Database

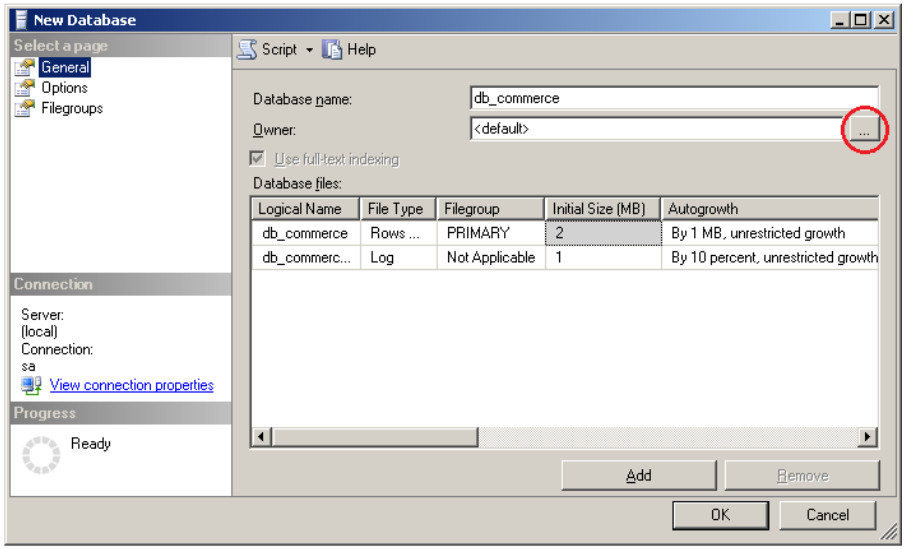
Untuk membuat *database* di SQL Server dapat digunakan dua cara, yaitu dengan mengetik perintah SQL langsung melalui jendela **Query Editor** yang tersedia atau dengan SSMS berbasis GUI. Berikut adalah perintah SQL yang digunakan untuk membuat *database* baru lengkap dengan berbagai pengaturannya.

```
USE [master]
GO
/****** Object:  Database [db_commerce]
        Script Date: 12/24/2013 20:23:30 ******/
CREATE DATABASE [db commerce] ON  PRIMARY
( NAME = N'db_commerce', FILENAME = N'C:\Program Files\Microsoft
SQL   Server\MSSQL10_50.MSSQLSERVER\MSSQL\DATA\db_commerce.mdf'   ,
SIZE = 2048KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
 LOG ON
(   NAME   =   N'db commerce log',   FILENAME   =   N'C:\Program
Files\Microsoft                                              SQL
Server\MSSQL10_50.MSSQLSERVER\MSSQL\DATA\db_commerce_log.ldf'     ,
SIZE = 1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO
ALTER DATABASE [db commerce] SET COMPATIBILITY LEVEL = 100
GO
IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [db_commerce].[dbo].[sp_fulltext_database] @action = 'enable'
end
GO
ALTER DATABASE [db commerce] SET ANSI NULL DEFAULT OFF
GO
ALTER DATABASE [db_commerce] SET ANSI_NULLS OFF
GO
ALTER DATABASE [db_commerce] SET ANSI_PADDING OFF
GO
ALTER DATABASE [db commerce] SET ANSI WARNINGS OFF
GO
ALTER DATABASE [db_commerce] SET ARITHABORT OFF
GO
ALTER DATABASE [db_commerce] SET AUTO_CLOSE OFF
GO
ALTER DATABASE [db_commerce] SET AUTO_CREATE_STATISTICS ON
GO
ALTER DATABASE [db_commerce] SET AUTO_SHRINK OFF
GO
ALTER DATABASE [db commerce] SET AUTO UPDATE STATISTICS ON
GO
ALTER DATABASE [db commerce] SET CURSOR CLOSE ON COMMIT OFF
GO
ALTER DATABASE [db_commerce] SET CURSOR_DEFAULT  GLOBAL
GO
ALTER DATABASE [db commerce] SET CONCAT NULL YIELDS NULL OFF
GO
ALTER DATABASE [db commerce] SET NUMERIC ROUNDABORT OFF
GO
ALTER DATABASE [db_commerce] SET QUOTED_IDENTIFIER OFF
GO
ALTER DATABASE [db commerce] SET RECURSIVE TRIGGERS OFF
GO
ALTER DATABASE [db_commerce] SET  DISABLE_BROKER
GO
ALTER DATABASE [db_commerce] SET AUTO_UPDATE_STATISTICS_ASYNC OFF
GO
```

```
ALTER DATABASE [db commerce] SET DATE CORRELATION OPTIMIZATION OFF
GO
ALTER DATABASE [db commerce] SET TRUSTWORTHY OFF
GO
ALTER DATABASE [db_commerce] SET ALLOW_SNAPSHOT_ISOLATION OFF
GO
ALTER DATABASE [db_commerce] SET PARAMETERIZATION SIMPLE
GO
ALTER DATABASE [db commerce] SET READ COMMITTED SNAPSHOT OFF
GO
ALTER DATABASE [db_commerce] SET HONOR_BROKER_PRIORITY OFF
GO
ALTER DATABASE [db commerce] SET  READ WRITE
GO
ALTER DATABASE [db commerce] SET RECOVERY SIMPLE
GO
ALTER DATABASE [db_commerce] SET  MULTI_USER
GO
ALTER DATABASE [db commerce] SET PAGE VERIFY CHECKSUM
GO
ALTER DATABASE [db_commerce] SET DB_CHAINING OFF
GO
```

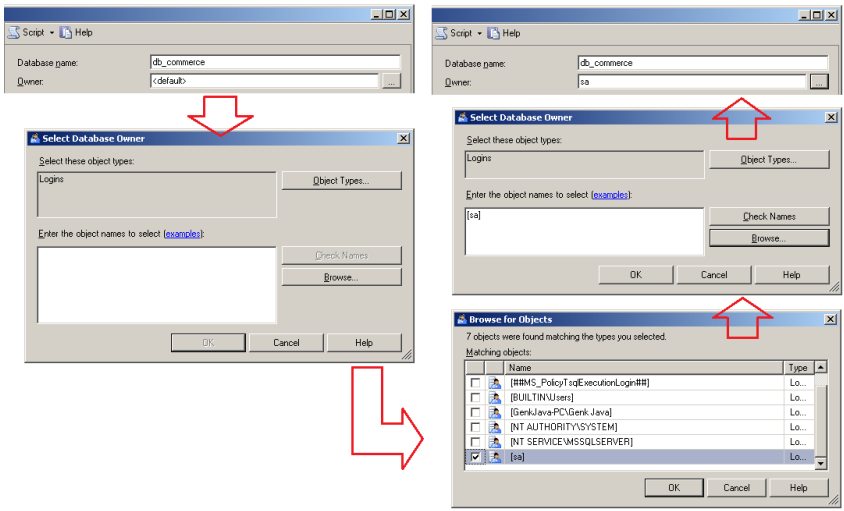
Sedangkan, berikut ini merupakan langkah-langkah yang perlu dilakukan dalam membuat sebuah *database* baru menggunakan aplikasi *wizard* berbasis GUI dengan SSMS.

1) Setelah berhasil masuk ke aplikas SSMS, pada bagian **Object Explorer**, klik kanan **Databases » New Database...**.
2) Setelah muncul jendela **New Database**, ketik "**db_commerce**" pada kolom **Database name**.
3) Klik tombol **telusur** (*browse* – tombol dengan label tiga titik "**...**") yang ada di sebelah kanan kolom **Owner**, seperti Gambar 5.1.



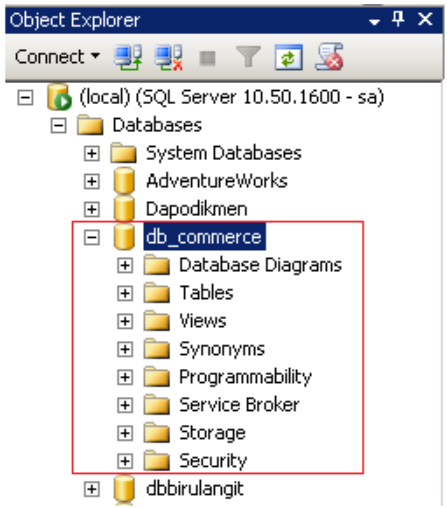
Gambar 5.1: Bagian **Object Explorer** dan jendela **New Database**

4) Sampai muncul jendela **Select Database Owner**, klik tombol **[Browse...]**.
5) Pada jendela **Browse for Objects**, beri tanda centang objek **[sa]** pada daftar **Matching object** dan klik tombol **[OK]**, maka akan dibawa kembali ke jendela **Select Database Owner**, klik tombol **[OK]** lagi.
6) Apabila pada kolom **Owner** sudah berisi objek **[sa]**, berarti sudah benar.



Gambar 5.2: Proses memasukkan objek **Owner**

7) Setelah kembali ke jendela **New Database** (Gambar 5.1), klik tombol **[OK]**. Jika berhasil, maka *database* "**db_commerce**" akan muncul di bagian **Object Explorer**, tepatnya di dalam grup **Databases** seperti Gambar 5.3.



Gambar 5.3: *Database* "**db_commerce**" yang baru dibuat

5.2 Membuat Tabel

Setelah membuat *database*, langkah berikutnya adalah membuat tabel. Tabel merupakan komponen utama *database* relasional yang digunakan untuk menyimpan data dalam bentuk kolom (*field*) dan baris (*record*).

5.2.1 Tabel [dbo].[products]

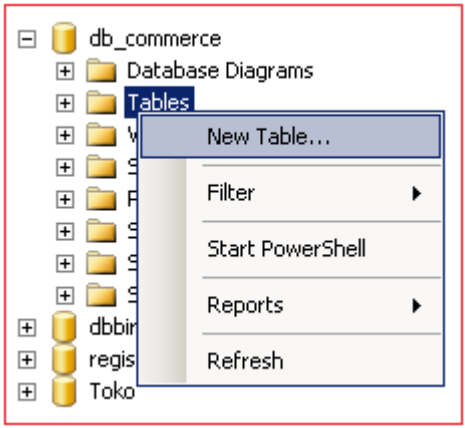
Tabel pertama yang akan dibuat adalah "**dbo.products**". Tabel ini digunakan untuk menyimpan data produk atau komoditas jual. Keterangan dari tabel "**dbo.products**" dituangkan dalam Tabel 5.1.

Tabel 5.1: Keterangan Tabel "**dbo.products**"

No.	Nama Kolom	Keterangan
1.	serial	Digunakan untuk menyimpan data ID-Produk. Kolom ini merupakan kunci primer (*primary key*) pada tabel **products**. Bertipe data **int** dengan atribut **Is Identify=Yes**, sehingga tanpa harus menentukan nilai (*value*), kolom ini akan terisi secara otomatis dengan nilai lebih besar satu angka dari baris data sebelumnya.
2.	name	Digunakan untuk menyimpan data nama produk.
3.	description	Digunakan untuk menyimpan data deskripsi suatu produk, mulai dari merk, spesifikasi, garansi, dan keterangan lain yang dianggap perlu ditambahkan.
4.	price	Digunakan untuk menyimpan data harga jual suatu produk.
5.	picture	Digunakan untuk menyimpan data lokasi file (*path*) gambar suatu produk disimpan.
6.	stock	Digunakan untuk menyimpan data stok suatu produk.

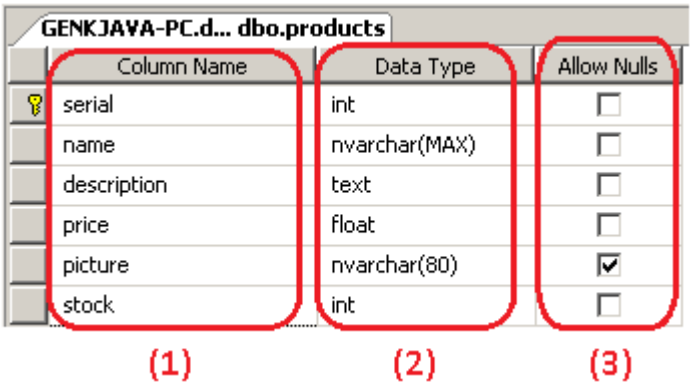
Adapun langkah-langkah membuat tabel "**dbo.products**" menggunakan SSMS adalah sebagai berikut.

1) Pilih *database* "**db_commerce**", kemudian klik kanan (*right clicked*) objek **Tables » New Table...**.



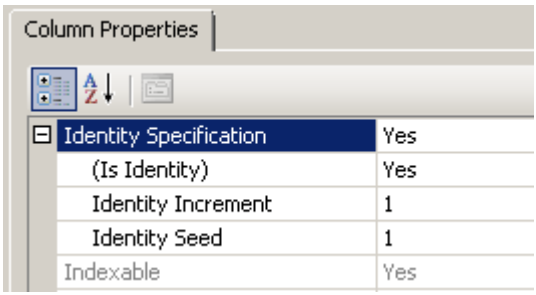
Gambar 5.4: Proses membuat tabel baru

2) Ketik nama kolom (*field*) yang diperlukan melalui bagian **Column Name**, tentukan tipe data kolom melalui bagian **Data Type**, dan status boleh kosong (*null*) suatu kolom melalui bagian **Allow Nulls** seperti Gambar 5.5 berikut ini.



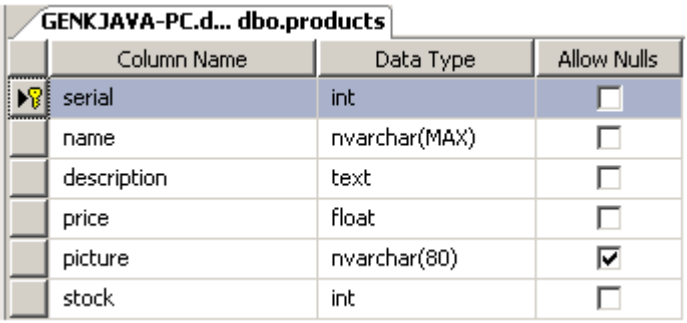
Gambar 5.5: Proses pembuatan tabel **[dbo].[products]**

3) Khusus untuk kolom **serial**, karena akan dibuat *auto-number* atau *auto-increment*, maka perlu dilakukan pengaturan lebih lanjut, yaitu dengan mengatur atribut **Is Identity** dengan nilai "**Yes**" dan **Identity Increment** dengan nilai "**1**".



Gambar 5.6 Properti untuk kolom **serial** pada tabel **[dbo].[products]**

4) Simpan tabel dengan mengklik ikon disket (tombol simpan) yang ada di bagian *toolbar* SSMS dan beri nama "**products**", maka secara otomatis akan bernama "**dbo.products**".



Gambar 5.7: Struktur akhir tabel **[dbo].[products]**

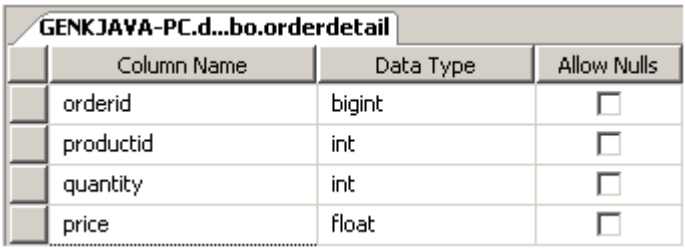
5.2.2 Tabel [dbo].[orderdetail]

Tabel "**dbo.orderdetail**" digunakan untuk menyimpan detail order atau pemesanan terhadap suatu produk. Tabel 5.2 menunjukkan keterangan dari tabel "**dbo.orderdetail**".

Tabel 5.2: Keterangan Tabel "**dbo.orderdetail**"

No.	Nama Kolom	Keterangan
1.	orderid	Digunakan untuk menyimpan data ID-Order. Kolom ini merupakan kunci asing (*foreign key*) yang mengacu pada kolom **serial** pada tabel "**dbo.orders**". Kolom ini bertipe data **bigint**, karena setiap kali order bisa lebih dari satu produk, sehingga di tabel "**dbo.orderdetail**" dimungkinkan akan menyimpan banyak *record*.
2.	productid	Digunakan untuk menyimpan data ID-Produk yang diorder. Kolom ini juga merupakan kunci asing (*foreign key*) yang mengacu pada kolom **serial** di tabel "**dbo.products**".
3.	quantity	Digunakan untuk menyimpan data kuantitas order suatu produk yang yang diorder.
4.	price	Digunakan untuk menyimpan data nilai order yang di dapat dari harga jual (**orderrdetail.price**) dikalikan dengan kuantitas (**orderdetail.quantity**). Kolom **orderdetail.price** ini bisa mempunyai nilai (*value*) yang sama dengan kolom **products.price**, tetapi juga bisa mempunyai nilai yang berbeda. Sama jika nilai yang tersimpan di kolom **products.price** belum mengalami perubahan (naik atau turun) sejak pertama kali dientri dan **orderdetail.quantity** bernilai **satu**, demikian sebaliknya.

Adapun struktur tabel "**dbo.orderdetail**" hasil pembuatan menggunakan SSMS ditunjukkan pada Gambar 5.8 berikut ini.



Gambar 5.8: Struktur tabel "**dbo.orderdetail**"

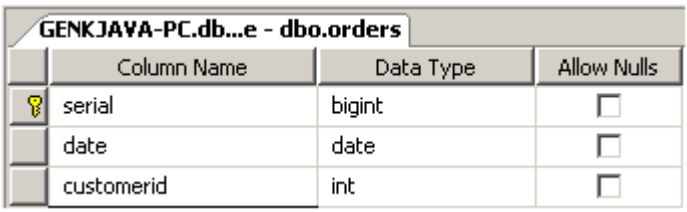
5.2.3 Tabel [dbo].[orders]

Tabel "**dbo.orders**" digunakan untuk menyimpan data order dari konsumen tertentu. Jika tabel "**dbo.orderdetail**" berhubungan dengan detail produk yang diorder, tabel "**dbo.orders**" berhubungan dengan data konsumen yang melakukan order. Secara teknis "**dbo.orderdetail**" merupakan tabel detail dari tabel "**dbo.orders**". Keterangan tabel "**dbo.orders**" ditunjukkan pada Tabel 5.3.

Tabel 5.3: Keterangan Tabel "**dbo.orders**"

No.	Nama Kolom	Keterangan
1.	serial	Digunakan untuk menyimpan data ID-Order. Kolom ini merupakan kunci primer (*primary key*) pada tabel ini. Kolom ini bertipe data **bigint** dengan atribut **Is Identify=Yes**.
2.	date	Digunakan untuk menyimpan data tanggal seorang konsumen melakukan order.
3.	customerid	Digunakan untuk menyimpan data ID-Konsumen yang melakukan order.

Sruktur tabel "**dbo.orders**" hasil pembuatan menggunakan SSMS ditunjukkan pada Gambar 5.9.



Gambar 5.9: Struktur tabel "**dbo.orderdetail**"

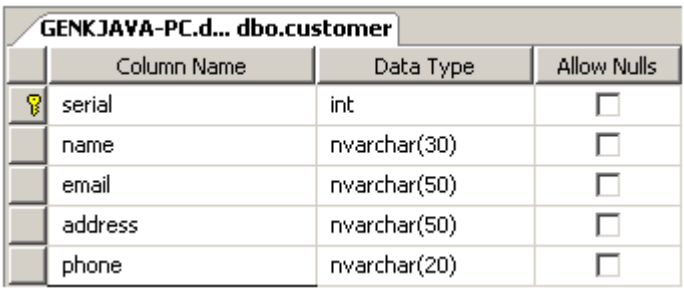
5.2.4 Tabel [dbo].[customer]

Table "**dbo.customer**" digunakan untuk menyimpan data konsumen yang pernah melakukan order. Keterangan dari tabel "**dbo.customer**" dituangkan dalam Tabel 5.4.

Tabel 5.4: Keterangan Tabel "**dbo.customer**"

No.	Nama Kolom	Keterangan
1.	serial	Digunakan untuk menyimpan data ID-Konsumen. Kolom ini merupakan kunci primer (*primary key*). Kolom ini juga bertipe data **bigint** dengan atribut **Is Identify=Yes**.
2.	name	Digunakan untuk menyimpan data nama konsumen.
3.	email	Digunakan untuk menyimpan data e-mail konsumen.
4.	address	Digunakan untuk menyimpan data alamat konsumen, yang selanjutnya digunakan sebagai alamat pengiriman barang yang diorder, tentunya setelah konsumen mentransfer sejumlah uang sesuai nilai ordernya.
5.	phone	Digunakan untuk menyimpan data no. telepon konsumen.

Gambar 5.10 berikut ini menunjukkan struktur tabel "**dbo.customer**" hasil pembuatan menggunakan SSMS.



Gambar 5.10: Struktur tabel "**dbo.customer**"

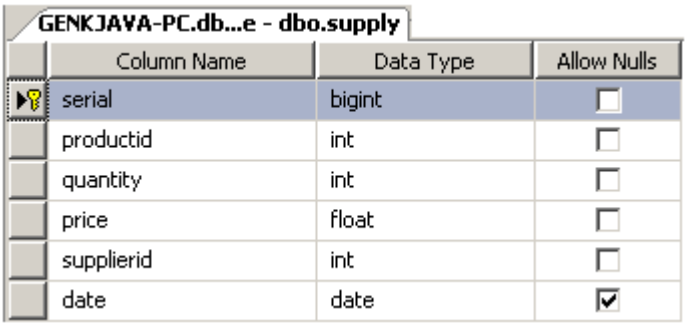
5.2.5 Tabel [dbo].[supply]

Tabel "**dbo.supply**" digunakan untuk menyimpan data pembelian/kulakan suatu produk. Keterangan dari tabel "**dbo.supply**" dituangkan dalam Tabel 5.5.

Tabel 5.5: Keterangan Tabel "**dbo.supply**"

No.	Nama Kolom	Keterangan
1.	serial	Digunakan untuk menyimpan data ID-Pembelian. Kolom ini merupakan kunci primer (*primary key*). Kolom ini juga bertipe data **bigint** dengan atribut **Is Identify=Yes**.
2.	productid	Digunakan untuk menyimpan data ID-Produk yang diorder. Kolom ini merupakan kunci asing (*foreign key*) yang mengacu pada kolom **serial** di tabel "**dbo.products**".
3.	quantity	Digunakan untuk menyimpan data kuantitas produk yang dibeli.
4.	price	Digunakan untuk menyimpan data harga satuan pembelian sutu produk.
5.	supplierid	Digunakan untuk menyimpan data ID-Supplier yang mnyuplai barang. Kolom ini juga merupakan kunci asing (*foreign key*) yang mengacu pada kolom **serial** di tabel "**dbo.supplier**".
6.	date	Digunakan untuk menyimpan data tanggal pembelian produk tertentu.

Gambar 5.11 berikut ini menunjukkan struktur tabel "**dbo.supply**" hasil pembuatan menggunakan SSMS.



Gambar 5.11: Struktur tabel "**dbo.supply**"

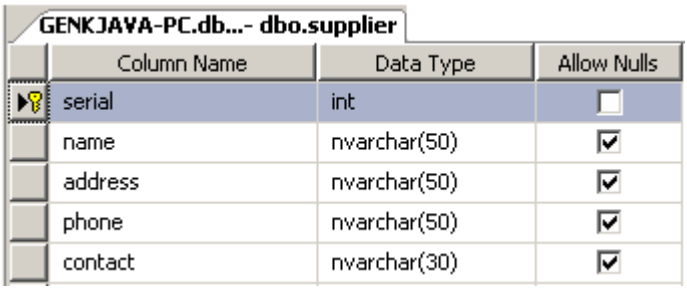
5.2.6 Tabel [dbo].[supplier]

Tabel "**dbo.supplier**" digunakan untuk menyimpan data penyedia produk (*supplier*). Keterangan dari tabel "**dbo.supplier**" dituangkan dalam Tabel 5.6 berikut ini.

Tabel 5.6: Keterangan Tabel "**dbo.supplier**"

No.	Nama Kolom	Keterangan
1.	serial	Digunakan untuk menyimpan data ID-Supplier. Kolom ini merupakan kunci primer (*primary key*). Kolom ini juga bertipe data **int** dengan atribut **Is Identify=Yes**.
2.	name	Digunakan untuk menyimpan data nama *supplier* yang menyediakan produk tertentu.
3.	address	Digunakan untuk menyimpan data alamat *supplier*.
4.	phone	Digunakan untuk menyimpan data no. Telepon atau HP *supplier* tertentu.
5.	contact	Digunakan untuk menyimpan data kontak person *supplier* tertentu.

Gambar 5.12 berikut ini menunjukkan struktur tabel "**dbo.supplier**" hasil pembuatan menggunakan SSMS.



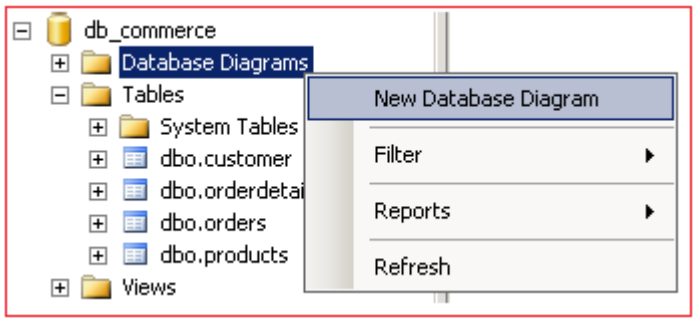
Gambar 5.12: Struktur tabel "**dbo.supplier**"

5.3 Membuat Diagram Database

Diagram *database* digunakan untuk menampilkan relasi antar tabel yang ada di dalam suatu *database*. Dengan adanya diagram ini,

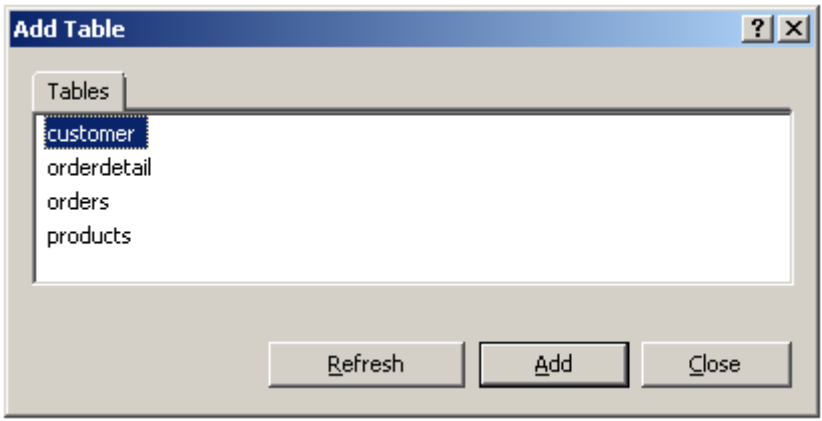
informasi dependensi antara data di satu tabel dengan tabel lainnya akan semakin jelas. Diagram ini juga sangat bermanfaat dalam membuat *view* kategori kompleks, perancang *database* akan dengan mudah melibatkan tabel mana saja dalam membuat sebuah *view*. Tentunya, masih banyak lagi manfaat yang diperoleh dengan adanya sebuah diagram *database*. Berikut ini disajikan langkah-langkah membuat diagram *database* menggunakan SSMS.

1) Pada bagian **Object Explorer**, pilih **Database Diagram**, kemudian pilih menu **New Database Diagram**.



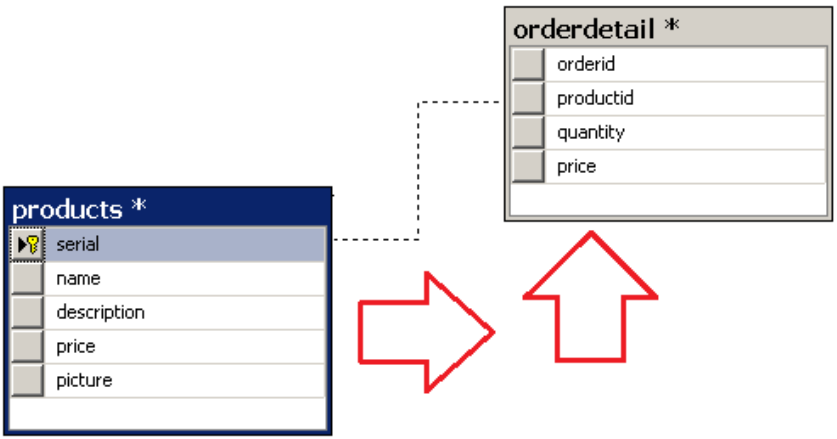
Gambar 5.13: Proses pembuatan diagram *database*

2) Setelah muncul jendela **Add Table**, pilih semua tabel yang ada, kemudian klik tombol **[Add]**.



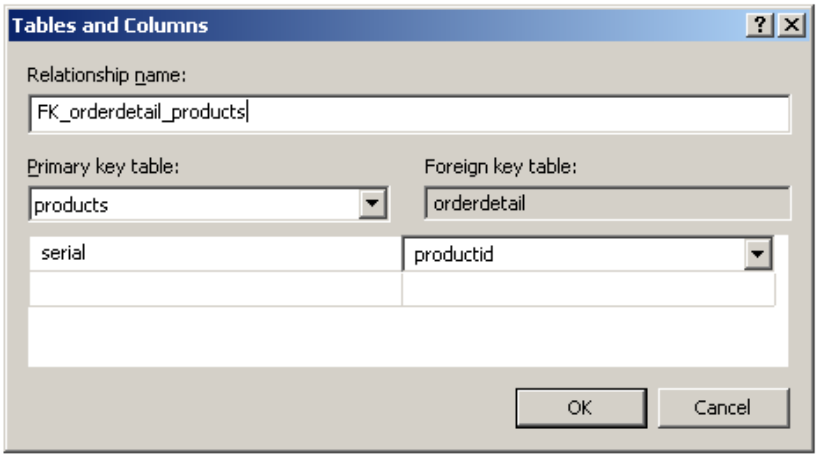
Gambar 5.14: Jendela **Add table**

3) Jika tidak terjadi kesalahan, maka keempat tabel yang dipilih tadi akan muncul ke dalam kanvas **Database Diagram**.



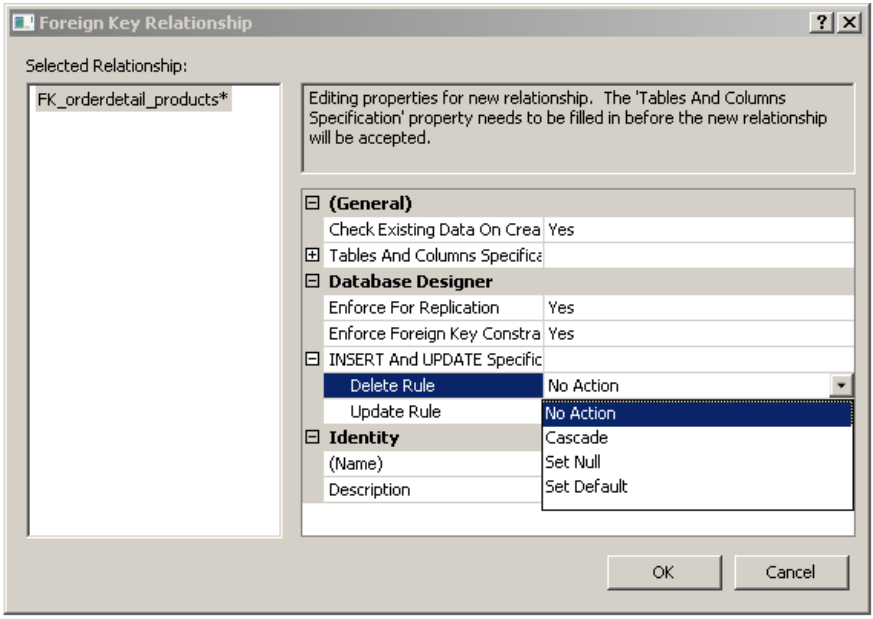
Gambar 5.15: Proses merelasikan tabel **products** dengan **orderdetail**

4) Untuk merelasikan antara tabel satu dengan lainnya, cukup *drag-and-drop* (klik tahan dan geser) kolom yang menjadi *primary key* ke kolom pada tabel lain yang menjadi *foreign key*–nya.
5) Setelah muncul jendela **Tables and Columns**, tentukan nama relasi melalui kolom **Relationship name**, nama tabel dan *primary key*-nya (sebelah kiri), serta nama tabel yang akan direlasikan dan *foreign key*-nya (sebelah kanan), kemudian klik tombol **[OK]**.



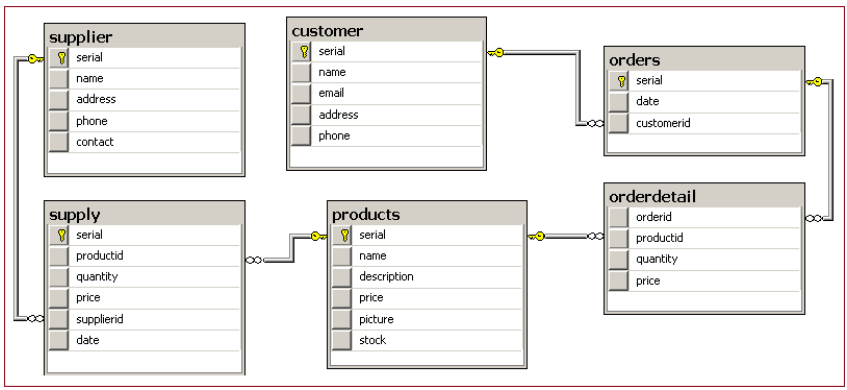
Gambar 5.16: Proses penentuan nama dan kunci relasinya

6) Jendela berikutnya adalah **Foreign Key Relationship**. Pada jendela ini, Anda diminta untuk menentukan aturan-aturan mengenai tabel yang mempunyai *foreign key* (tabel **orderdetail**).



Gambar 5.17: Jendela **Foreign Key Relationship**

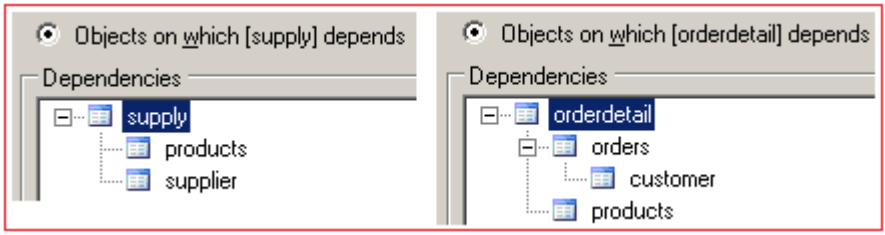
7) Aturan-aturan yang dimaksud diantaranya adalah **INSERT and UPDATE Specification**. Pada aturan ini, baik **Delete Rule** maupun **Update Rule** sama-sama mempunyai empat opsi, yaitu:
   - **No Action**. Jika terjadi proses *update* atau *delete* pada data di tabel yang mempunyai *primary key* (tabel **products**), maka data-data yang bersesuaian di tabel yang mempunyai *foreign key* (tabel **orderdetail**) tidak akan terjadi apap-apa.
   - **Cascade**. Jika terjadi proses *update* atau *delete* pada data di tabel **products**, maka data-data yang bersesuaian di tabel **orderdetail** juga akan terbarui (*updated*) atau terhapus (*deleted*).
   - **Set Null**. Jika terjadi proses *update* atau *delete* pada data di tabel **products**, maka data-data yang bersesuaian di tabel **orderdetail** akan di set menjadi kosong (bernilai *null*).
   - **Set Default**. Jika terjadi proses *update* atau *delete* pada data di tabel **products**, maka data-data yang bersesuaian di tabel **orderdetail** akan dikembalikan ke nilai *default* yang ditentukan saat pembuatan tabel.
8) Dengan cara yang sama, relasikan **supplier.serial** dengan **supply.supplierid**, **products.serial** dengan **supply.productid**, **products.serial** dengan **orderdetail.productid**, **orders.serial** dengan **orderdetail.orderid**, dan yang terakhir **customer.serial** dengan **orders.customerid**. Gambar 5.18 berikut merupakan diagram *database* yang telah dibuat.



Gambar 5.18: Diagram relasi antar tabel dalam *database* **db_commerce**

Dari diagram *database* di atas dapat disimpulkan beberapa hubungan dependensi sebagai berikut.

- Tabel **supply** tergantung (*depend*) pada tabel **supplier** dan tabel **products**.
- Tabel **orderdetail** tergantung pada tabel **orders** dan tabel **products**, sedangkan tabel **orders** tergantung pada tabel **customer**. Hubungan dependensi antar tabel ini ditunjukkan pada Gambar 5.19.



Gambar 5.19: Hubungan dependensi antar tabel dalam *database* **db_commerce**

5.4 Persiapan Data

Persiapan data merupakan proses memasukkan data ke dalam sebuah tabel sehingga tabel tersebut mempunyai data awal. Tujuannya agar efeknya lebih representatif ketika sebuah tabel tadi dikenai suatu perintah SQL. Berikut ini disajikan beberapa perintah SQL untuk menyisipkan (*insert*) dan menampilkan (*select*) data ke/dari dalam sebuah tabel.

- Menyisipkan tujuh data ke dalam tabel **products**.

```
INSERT [dbo].[products] (name, description, price, picture, stock) VALUES
('Samsung Galaxy Tab 3', 'Garansi 1 Tahun.', 5250000, 'images/mobile.jpg', 100),
('Atech Mouse', 'Garansi 1,5 Bulan.', 68750, 'images/mouse.jpg', 100),
('Flassdisk 8 GB Kingston', N'Garansi 10 Bulan.', 79900, 'images/flash-disk.jpg', 100),
('Laptop Charger', 'Garansi 6 Bulan.', 569700, 'images/dell-charger.jpg', 100),
('LG LED Monitor 20"', 'Garansi 11 Bulan.', 1250000, 'images/lcd.jpg', 100),
('LG DVD-ROM Drive', 'Garansi 1 Tahun', 90500, 'images/cdrom-drive.jpg', 100),
('Seagate Harddisk 1 TB', 'Garansi 2 Tahun', 650000, 'images/hard-drive.jpg', 100);

Messages
(7 row(s) affected)
```

Gambar 5.20: Proses *insert* data ke dalam tabel **dbo.products**

- Dengan perintah SQL serupa Anda dapat bereksperimen sendiri untuk menyisipkan data ke dalam tabel **customer**, **orders** dan **orderdetail**.
- Menampilkan seluruh data yang ada dalam tabel **products**.

```
SELECT * FROM [db_commerce].[dbo].[products]
```

Results | Messages

	serial	name	description	price	picture	stock
1	48	Samsung Galaxy Tab 3	Garansi 1 Tahun.	5250000	images/mobile.jpg	100
2	49	Atech Mouse	Garansi 1,5 Bulan.	68750	images/mouse.jpg	100
3	50	Flassdisk 8 GB Kingston	Garansi 10 Bulan.	79900	images/flash-disk.jpg	100
4	51	Laptop Charger	Garansi 6 Bulan.	569700	images/dell-charger.jpg	100
5	52	LG LED Monitor 20"	Garansi 11 Bulan.	1250000	images/lcd.jpg	100
6	53	LG DVD-ROM Drive	Garansi 1 Tahun	90500	images/cdrom-drive.jpg	100
7	54	Seagate Harddisk 1 TB	Garansi 2 Tahun	650000	images/hard-drive.jpg	100

Gambar 5.21: Proses menampilkan data pada tabel **dbo.products**

- Menampilkan seluruh data yang ada dalam tabel **customer**.

```
SELECT [serial]
      ,[name]
      ,[email]
      ,[address]
      ,[phone]
  FROM [db_commerce].[dbo].[customer]
```

Results | Messages

	serial	name	email	address	phone
1	1	Mariya Ulfa	mariya@gmail.com	Pati	081328[redacted]
2	8	Garda Revolusi	garda@gmail.com	Ngalangan	085292[redacted]

Gambar 5.22: Proses menampilkan data pada tabel **dbo.customer**

- Menampilkan seluruh data yang ada dalam tabel **orders**.

```
SELECT [serial]
      ,[date]
      ,[customerid]
  FROM [db_commerce].[dbo].[orders]
```

Results | Messages

	serial	date	customerid
1	8	2013-10-23	8

Gambar 5.23: Proses menampilkan data pada tabel **dbo.orders**

- Menampilkan seluruh data yang ada dalam tabel **orderdetail**.

```
SELECT [orderid]
      ,[productid]
      ,[quantity]
      ,[price]
  FROM [db_commerce].[dbo].[orderdetail]
```

Results | Messages

	orderid	productid	quantity	price
1	8	30	1	5250000
2	8	34	1	1250000
3	8	32	2	79900

Gambar 5.24: Proses menampilkan data pada tabel **dbo.orderdetail**

- Menampilkan seluruh data yang ada dalam tabel **supplier**.

```
SELECT TOP 1000 [serial]
      ,[name]
      ,[address]
      ,[phone]
      ,[contact]
  FROM [db_commerce].[dbo].[supplier]
```

Results | Messages

	serial	name	address	phone	contact
1	1	PT. Mutiara Teknologi	Jl. Gejayan No. 456A	0274-9999999	Ngatio, M.Sc.
2	2	PT. Gampang	Jl. Kaliurang Km. 10	0274-8888888	Dra. Suginem
3	3	PT. Maju Mundur Maju	Jl. Magelang Km.12	0274-7777777	Drs. Heru
4	4	PT. Mundur Lancar	Jl. Monjali No. 189A	0274-6666666	Drs. Ramlan
5	5	PT. Maju Bisa Mundur Bisa	Jl. Lempongsari No. 555	0274-5555555	Dra. Buyut Warni

Gambar 5.25: Proses menampilkan data pada tabel **dbo.supplier**

- Menampilkan seluruh data yang ada dalam tabel **supply**.

```
SELECT TOP 1000 [serial]
      ,[productid]
      ,[quantity]
      ,[price]
      ,[supplierid]
      ,[date]
  FROM [db_commerce].[dbo].[supply]
```

Results | Messages

	serial	productid	quantity	price	supplierid	date
1	2	48	10	5000000	1	2014-02-25
2	3	52	10	1050000	1	2014-02-25
3	4	54	10	575000	1	2014-02-25

Gambar 5.26: Proses menampilkan data pada tabel **dbo.supply**

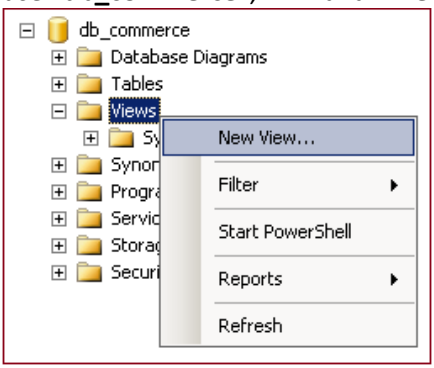
5.5 Membuat View

View juga termasuk dalam komponen *database*. Secara *default*, *view* baru dibuat ke dalam *database* yang diaktifkan. Untuk membuat

secara eksplisit di dalam suatu *database* tertentu, maka buatlah nama *view* dengan sintaks: **db_name.view_name**.

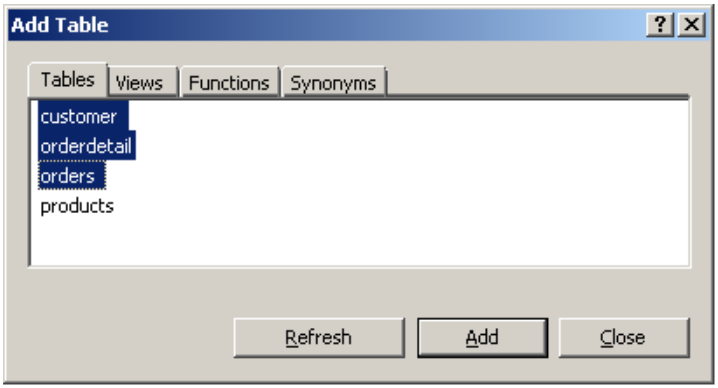
Untuk membuat *view*, Anda bisa menggunakan dua cara, yaitu: menggunakan **WIZARD** dan **CODING** (mengetik langsung) ke editor *query*. Pada bab ini akan dibahas pembuatan *view* menggunakan **WIZARD** karena untuk membuat *view* melalui **CODING**, Anda cukup mengetikkan perintah SQL ke dalam editor *query* kemudian mengeksekusinya.

1) Pilih *database* "**db_commerce**", klik kanan **Views » New View...**.



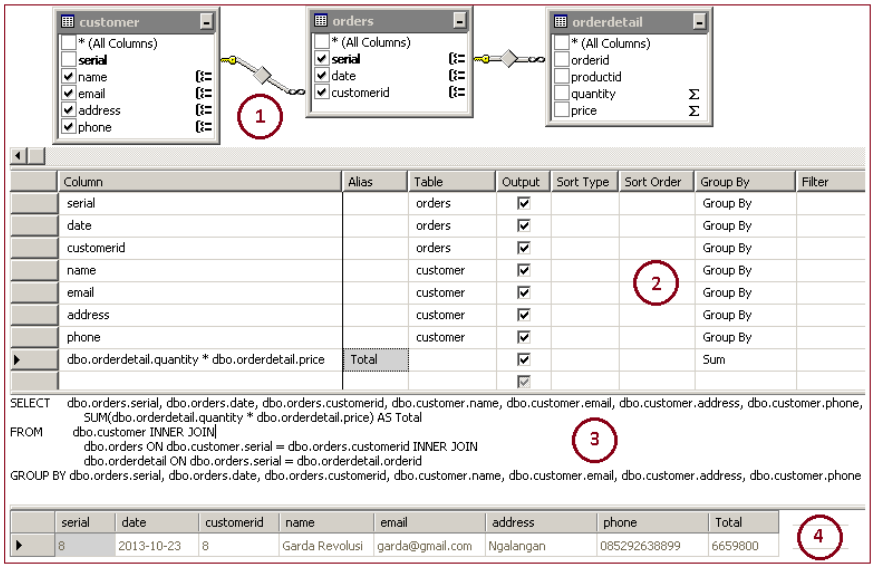
Gambar 5.27: Proses pembuatan *view* baru

2) Setelah muncul kotak dialog **Add Table**, pilih tabel apa saja yang ingin dilibatkan, kemudian klik tombol **[Add]**.



Gambar 5.28: Kotak dialog **Add table**

3) Anda akan dibawa ke halaman utama pembuatan *view*. Halaman utama *view* ditunjukkan pada Gambar 5.29.

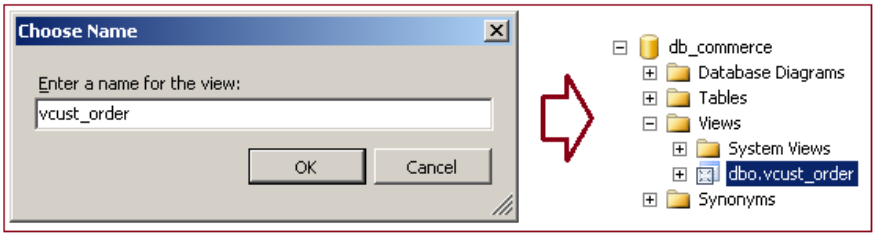


Gambar 5.29: Proses pembuatan *view*

Keterangan:

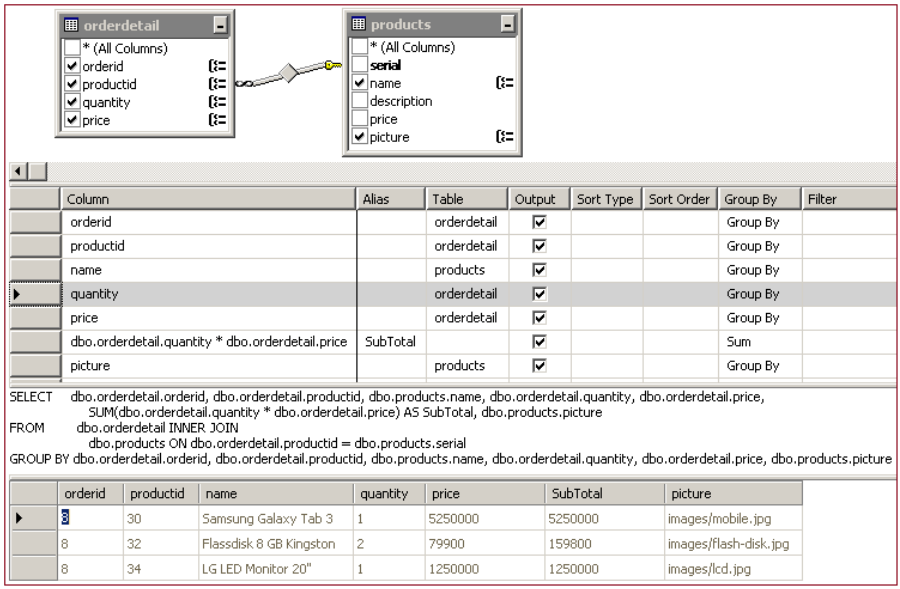
- [1] Daftar tabel yang ditambahkan ke dalam pembuatan *view*.
- [2] *View worksheet* yang digunakan untuk menentukan kolom, alias, nama tabel, status sebagai *output*, tipe pengurutan, status pengurutan, status pengegrupan, filterisasi dan lainnya.
- [3] Kode hasil generate dari *view worksheet*.
- [4] Hasil eksekusi *view* berdasarkan atribut yang ditentukan di *view worksheet*.

4) Untuk menampilkan hasil dari *view* yang dibuat, klik tombol  (tanda seru) yang ada di *toolbar*.
5) Untuk menyimpan, klik tombol disket yang ada di *toolbar*, setelah muncul kotak dialog **Choose Name**, klik tombol **[OK]**.



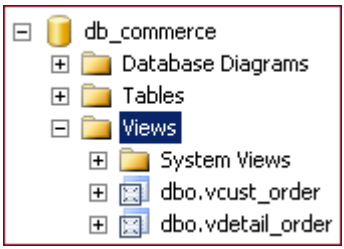
Gambar 5.30: Proses penyimpanan *view*

6) Dengan cara yang sama, buat *view* satu lagi dengan deskripsi yang ditunjukkan pada Gambar 5.31 dan simpan dengan nama "**vdetail_order**".



Gambar 5.31: Proses pembuatan *view* "**vdetail_order**"

7) Jika tidak ada kesalahan, maka di dalam daftar *view* akan muncul nama "**vdetail_order**" seperti Gambar 5.32 berikut ini.



Gambar 5.32: Daftar *view* yang telah dibuat

5.6 Membuat Stored Procedure

Stored Procedure merupakan kumpulan perintah SQL yang didefinisikan oleh pengguna dan disimpan dengan nama tertentu di dalam server *database*. *Stored Procedure* biasanya berisi perintah-perintah umum yang berhubungan dengan *database*, baik perintah DDL (*data definition language*) maupun DML (*data manipulation language*). Berikut ini disajikan beberapa contoh pembuatan *stored procedure* untuk menampilkan (*select*), menyisipkan (*insert*), memperbarui (*update*) dan menghapus (*delete*) data. Secara umum sintaks *Stored Procedure* sebagai berikut.

```
CREATE PROCEDURE [schema_name.][Procedure_Name]
       <@Param_1> <Datatype_For_Param_1> =
<Default_Value_For_Param_1>,
       <@Param_2> <Datatype_For_Param_2> =
<Default_Value_For_Param_2>,
        ...
       <@Param_n> <Datatype_For_Param_n> =
<Default_Value_For_Param_n>
AS
       SQL_STATEMENT;
```

Tabel 5.7: Keterangan sintaks *stored procedure*

Argumen	Keterangan
`[schema_name.][Procedure_Name]`	Mendeskripsikan nama skema dan nama *stored procedure*.
`<Param> <Datatype_For_Param> = <Default_Value_For_Param>`	Mendeskripsikan nama parameter, tipe data parameter, dan nilai *default* untuk parameter. Nama parameter harus diawali dengan tanda "**@**".

	Parameter ini bersifat opsional, bisa terdiri satu atau lebih parameter, tetapi juga tidak harus ada.
SQL_STATEMENT	Merupakan isi dari *stored procedure* yang berisi kumpulan perintah SQL.

Mungkin Anda bertanya, mengapa diperlukan *stored procedure*? Berikut disajikan kelebihan dan kekurangan penerapan *stored procedure* dalam sistem *database*. Terdapat banyak kelebihan diterapkan *stored procedure*, diantaranya adalah sebagai berikut.

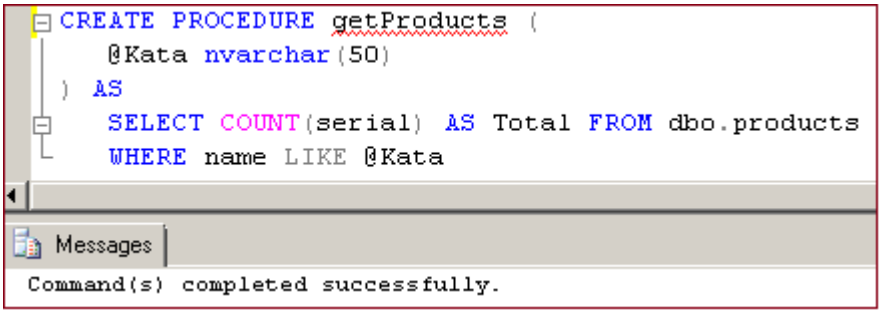
- Karena bisa menerapkan parameter, *stored procedure* lebih fleksibel dalam melakukan berbagai operasi data.
- Proses eksekusi *stored procedure* dilakukan di server *database* sehingga prosesnya menjadi lebih cepat.
- Lebih singkat kode program dalam mengembangan aplikasi, karena aplikasi cukup memanggil *stored procedure* dan mengirim parameter yang diperlukan sesuai dengan format *store procedure* yang akan dieksekusi.
- Bisa digunakan untuk membagi beban kerja antara server dan klien saat aplikasi di jalankan. Jika semua *query* dibuat dan dijalankan sisi klien (program aplikasi atau *front end*), maka sumber daya yang terpakai pada sisi klien tersebut akan besar, dengan adanya *stored procedure* sebagian beban kerja klien bisa dialihkan ke server.
- *Stored procedure* hanya ditulis sekali, tetapi bisa diakses oleh banyak aplikasi.
- *Stored procedure* berparameter yang memvalidasi seluruh input *user* dapat digunakan untuk mencegah terjadinya SQL *injection*. Jika Anda menggunakan perintah SQL dinamis, pastikan perintah Anda berparameter, dan jangan pernah memasukkan nilai parameter secara langsung ke dalam perintah *query*.
- *Stored procedure* dapat mereduksi lalu lintas jaringan, yaitu dengan mengombinasikan banyak operasi data ke dalam satu pemanggilan *procedure*.

Sedangkan, kekurangan dalam penerapan *stored procedure* diantaranya adalah sebagai berikut.

- Apabila semua permintaan operasi data ditangani oleh server *database* menggunakan *stored procedure*, jika jumlah kliennya sangat banyak, maka beban server akan sangat besar.
- Kesulitan dalam melakukan migrasi server *database*, misalkan dari SQL Server ke Oracle, SQL Server ke MySQL, atau sebaliknya. Karena sintaks *stored procedure* pada umumnya berbeda pada masing-masing DBMS..

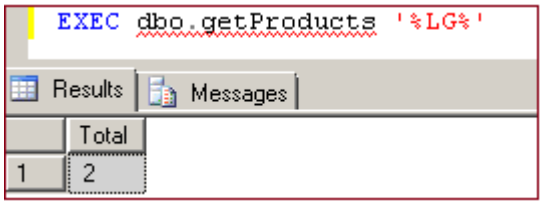
5.6.1 Stored Procedure untuk Menampilkan Data

Berikut ini disajikan contoh *stored procedure* untuk menampilkan data pada tabel **dbo.products** dengan kriteria tertentu.



Gambar 5.33: *Stored procedure* "**getProducts**"

Stored procedure "**getProducts**" digunakan untuk menampilkan jumlah data (hasilnya ditampilkan dalam kolom "**Total**") pada tabel **dbo.products** dengan kriteria nama produk (kolom "**name**") mengandung kata dalam varibel **@Kata**.



Gambar 5.34: Proses dan hasil eksekusi *procedure* "**getProducts**"

Jika *stored procedure* "**getProducts**" menampilkan data dari **tabel**, berikut ini disajikan contoh satu lagi *stored procedure* untuk menampilkan data dari **view**.

```
CREATE PROCEDURE getDetailOrder (
@id bigint
) AS
SELECT * FROM vdetail_order WHERE orderid=@id;
```

Stored procedure "**getDetailOrder**" digunakan untuk menampilkan data dari *view* "**vdetail_order**" dengan kriteria ID-Order (kolom "**orderid**") bernilai varibel **@id**.

```
EXEC dbo.getDetailOrder '8';
```

Results | Messages

	orderid	productid	name	quantity	price	SubTotal	picture
1	8	30	Samsung Galaxy Tab 3	1	5250000	5250000	images/mobile.jpg
2	8	32	Flassdisk 8 GB Kingston	2	79900	159800	images/flash-disk.jpg
3	8	34	LG LED Monitor 20"	1	1250000	1250000	images/lcd.jpg

Gambar 5.35: Proses dan hasil eksekusi *procedure* "**getDetailOrder**"

5.6.2 Stored Procedure untuk Menyisipkan Data

Berikut ini disajikan contoh *stored procedure* untuk menyisipkan data ke dalam tabel **dbo.products**.

```
CREATE PROCEDURE insProducts(
    @vname nvarchar(MAX),
    @vdesc text,
    @vprice float,
    @vpic nvarchar(80)
) AS
INSERT INTO dbo.products(name, description, price, picture)
VALUES (@vname, @vdesc, @vprice, @vpic)
```

Messages

```
Command(s) completed successfully.
```

Gambar 5.36: *Stored procedure* "**insProducts**"

Stored procedure "**insProducts**" digunakan untuk menyisipkan data melalui empat variabel input, yaitu: **@vname** (untuk kolom "**name**"), **@vdesc** (untuk kolom "**description**"), **@vprice** (untuk kolom "**price**"), dan **@vpic** (untuk kolom "**picture**"). Kolom **dbo.products.serial** sengaja tidak didefinisikan karena sudah bertipe data **IDENTIFY (1,1)**, sehingga dapat terisi secara otomatis oleh sistem.

```
EXEC insProducts 'GADMEI TV Tunner USB Stick',
                 'Termasuk CD TVHome Media 3. Garansi 1 Tahun.',
                 '200000','image/gadmei-tv.jpg';
SELECT * FROM [db_commerce].[dbo].[products]
```

Results | Messages

	serial	name	description	price	picture
1	30	Samsung Galaxy Tab 3	Garansi 1 Tahun.	5250000	images/mobile.jpg
2	31	Atech Mouse	Garansi 1 Bulan.	62500	images/mouse.jpg
3	32	Flassdisk 8 GB Kingston	Garansi 10 Bulan.	79900	images/flash-disk.jpg
4	33	Laptop Charger	Garansi 6 Bulan.	569500	images/dell-charger.jpg
5	34	LG LED Monitor 20"	Garansi 11 Bulan.	1250000	images/lcd.jpg
6	35	LG DVD-ROM Drive	Garansi 1 Tahun	90500	images/cdrom-drive.jpg
7	36	Seagate Harddisk 1 TB	Garansi 2 Tahun	650000	images/hard-drive.jpg
8	37	GADMEI TV Tunner USB Stick	Termasuk CD TVHome Media 3. Garansi 1 Tahun.	200000	image/gadmei-tv.jpg

Gambar 5.37: Proses dan hasil eksekusi *procedure* "**insProducts**"

5.6.3 Stored Procedure untuk Memperbarui Data

Berikut ini disajikan contoh *stored procedure* untuk memperbarui data pada tabel **dbo.products**.

```
CREATE PROCEDURE updProducts(
    @id int,
    @vname nvarchar(MAX),
    @vdesc text,
    @vprice float,
    @vpic nvarchar(80)
) AS
UPDATE dbo.products SET name=@vname, description=@vdesc, price=@vprice, picture=@vpic
WHERE serial=@id
```

Messages

```
Command(s) completed successfully.
```

Gambar 5.38: *Stored procedure* "**updProducts**"

Stored procedure "**updProducts**" digunakan untuk memperbarui data melalui empat variabel edit dan satu variabel kondisi. Empat variabel edit yang dimaksud adalah **@vname** (untuk kolom "**name**"), **@vdesc** (untuk kolom "**description**"), **@vprice** (untuk kolom "**price**"),

dan **@vpic** (untuk kolom "**picture**"). Sedangkan variabel kondisnya **@id** (untuk kolom "**serial**").

```
EXEC updProducts '37', 'GADMEI TV Tunner USB Stick',
                 'Termasuk CD TVHome Media 3. Garansi 2 Tahun.',
                 '215000','image/gadmei-tv.jpg';
SELECT * FROM dbo.products;
```

Results | Messages

	serial	name	description	price	picture
1	30	Samsung Galaxy Tab 3	Garansi 1 Tahun.	5250000	images/mobile.jpg
2	31	Atech Mouse	Garansi 1 Bulan.	62500	images/mouse.jpg
3	32	Flassdisk 8 GB Kingston	Garansi 10 Bulan.	79900	images/flash-disk.jpg
4	33	Laptop Charger	Garansi 6 Bulan.	569500	images/dell-charger.jpg
5	34	LG LED Monitor 20"	Garansi 11 Bulan.	1250000	images/lcd.jpg
6	35	LG DVD-ROM Drive	Garansi 1 Tahun	90500	images/cdrom-drive.jpg
7	36	Seagate Harddisk 1 TB	Garansi 2 Tahun	650000	images/hard-drive.jpg
8	37	GADMEI TV Tunner USB Stick	Termasuk CD TVHome Media 3. Garansi 2 Tahun.	215000	image/gadmei-tv.jpg

Gambar 5.39: Proses dan hasil eksekusi *procedure* "**updProducts**"

5.6.4 Stored Procedure untuk Menghapus Data

Berikut ini disajikan contoh *stored procedure* untuk menghapus data dari tabel **dbo.products**.

```
CREATE PROCEDURE delProducts(
    @id int
) AS
DELETE FROM dbo.products WHERE serial=@id
```

Messages

```
Command(s) completed successfully.
```

Gambar 5.40: *Stored procedure* "**delProducts**"

Stored procedure "**delProducts**" digunakan untuk menghapus data dengan satu variabel kondisi, yaitu **@id** (untuk kolom "**serial**").

```
EXEC delProducts '37';
SELECT * FROM dbo.products;
```

Results | Messages

	serial	name	description	price	picture
1	30	Samsung Galaxy Tab 3	Garansi 1 Tahun.	5250000	images/mobile.jpg
2	31	Atech Mouse	Garansi 1 Bulan.	62500	images/mouse.jpg
3	32	Flassdisk 8 GB Kingston	Garansi 10 Bulan.	79900	images/flash-disk.jpg
4	33	Laptop Charger	Garansi 6 Bulan.	569500	images/dell-charger.jpg
5	34	LG LED Monitor 20''	Garansi 11 Bulan.	1250000	images/lcd.jpg
6	35	LG DVD-ROM Drive	Garansi 1 Tahun	90500	images/cdrom-drive.jpg
7	36	Seagate Harddisk 1 TB	Garansi 2 Tahun	650000	images/hard-drive.jpg

Gambar 5.41: Proses dan hasil eksekusi *procedure* "**delProducts**"

Dengan cara yang sama, ketik dan eksekusi skrip berikut ini untuk membuat *stored procedure* pada tabel **dbo.customer**.

```
-- INSERT DATA
CREATE PROCEDURE insCust(
   @vname nvarchar(30),
   @vmail nvarchar(50),
   @vaddr nvarchar(50),
   @vphone nvarchar(20)
) AS
INSERT INTO dbo.customer(name, email, address, phone)
VALUES (@vname, @vmail, @vaddr, @vphone);
GO
-- UPDATE DATA
CREATE PROCEDURE updCust(
   @id int,
   @vname nvarchar(30),
   @vmail nvarchar(50),
   @vaddr nvarchar(50),
   @vphone nvarchar(20)
) AS
UPDATE dbo.customer SET name=@vname, email=@vmail, address=@vaddr,
phone=@vphone
WHERE serial=@id;
GO
-- DELETE DATA
CREATE PROCEDURE delCust(
   @id int
) AS
DELETE FROM dbo.customer WHERE serial=@id;
GO
```

Selanjutnya, ketik dan eksekusi skrip berikut ini untuk membuat *stored procedure* pada tabel **dbo.orders**.

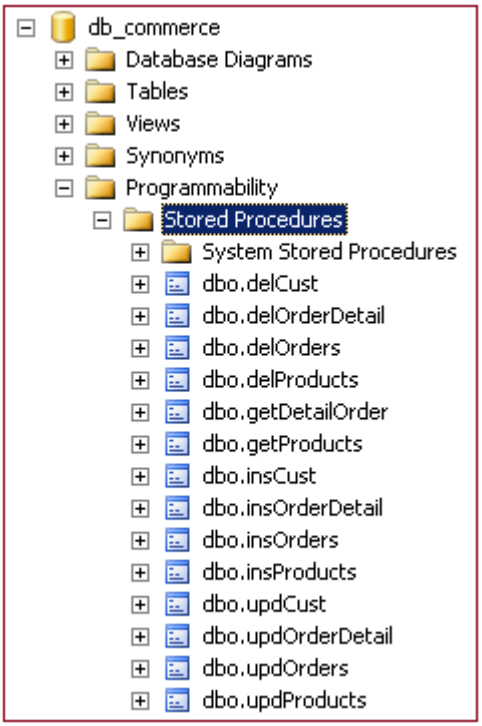
```
-- INSERT DATA
CREATE PROCEDURE insOrders(
   @vdate date,
   @vcust int
) AS
INSERT INTO dbo.orders (date, customerid) VALUES (@vdate,
@vcust);
GO
-- UPDATE DATA
CREATE PROCEDURE updOrders(
   @id bigint,
   @vdate date,
   @vcust int
) AS
UPDATE dbo.orders SET date=@vdate, customerid=@vcust
WHERE serial=@id;
GO
-- DELETE DATA
CREATE PROCEDURE delOrders(
   @id int
) AS
DELETE FROM dbo.orders WHERE serial=@id;
GO
```

Terakhir, ketik dan eksekusi skrip berikut ini untuk membuat *stored procedure* pada tabel **dbo.orderdetail**.

```
-- INSERT DATA
CREATE PROCEDURE insOrderDetail(
   @vorderid bigint,
   @vproid int,
   @vqty int,
   @vprice float
) AS
INSERT INTO dbo.orderdetail(orderid, productid, quantity,
price)
VALUES (@vorderid, @vproid, @vqty, @vprice);
GO
-- UPDATE DATA
CREATE PROCEDURE updOrderDetail(
   @vorderid bigint,
   @vproid int,
   @vqty int,
   @vprice float
```

```
) AS
UPDATE dbo.orderdetail SET quantity=@vqty, price=@vprice
WHERE orderid=@vorderid AND productid=@vproid;
GO
-- DELETE DATA
CREATE PROCEDURE delOrderDetail(
   @vorderid bigint,
   @vproid int
) AS
DELETE FROM dbo.orderdetail
WHERE orderid=@vorderid AND productid=@vproid;
GO
```

Gambar 5.**42** berikut ini menunjukkan daftar *stored procedure* yang telah dibuat pada *database* "**db_commerce**".



Gambar 5.42: Daftar *stored procedure* pada *database* "**db_commerce**"

5.7 Membuat UDF

Mungkin Anda sering melihat fungsi-fungsi SQL siap pakai bawaan sistem, seperti **MAX()** untuk menampilkan data terbesar (khusus untuk tipe **DATE** akan menampilkan data tanggal termuda) pada kolom tertentu, **SUM()** untuk menampilkan jumlah seluruh data pada kolom tertentu, **AVG()** untuk menampilkan rata-rata data yang ada di kolom tertentu, demikian fungsi-fungsi bawaan lainnya. Dengan **UDF** (*user defined function* – sering disebut **Function** saja), Anda dapat mendefinisikan sendiri fungsi-fungsi sesuai selera sehingga bisa digunakan selayaknya fungsi bawaan tersebut.

Pada SQL Server 2008 R2, UDF dikategorikan menjadi tiga, yaitu: **Inline Table-Valued Function**, **Scalar-Valued Function**, dan **Multi-Statement Table-Valued Function**.

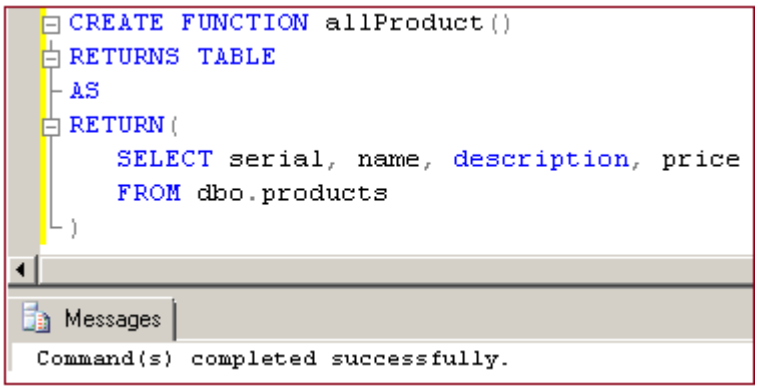
5.7.1 Inline Table-Valued Function

Inline table-valued function (disingkat ITVF) merupakan bagian dari UDF yang mengembalikan tipe data **table**. ITVF dapat digunakan untuk meningkatkan fungsionalitas *view* yang berparameter. Berikut ini adalah aturan-aturan dalam ITVF:

- Klausa **RETURN** hanya terdiri dari kata kunci **table**. Anda tidak harus mendefinisikan format dari variabel *return*, karena variabel tersebut diset oleh format *result set* yang dihasilkan dari statemen **SELECT** dalam klausa **RETURN**.
- **TIDAK ADA** *function body* yang diapit dengan **BEGIN** dan **END**.
- Klausa **RETURN** berisi sebuah statemen **SELECT** tunggal yang diapit tanda kurung. *Result set* yang dihasilkan oleh statemen **SELECT** membentuk **table** yang dikembalikan oleh fungsi. Statemen **SELECT** digunakan dalam sebuah fungsi *inline* mengikuti aturan statemen **SELECT** yang digunakan dalam *view*.
- ITVF hanya menerima konstanta atau argumen **@local_variable**. Skrip berikut ini merupakan sintaks umum dari ITVF.

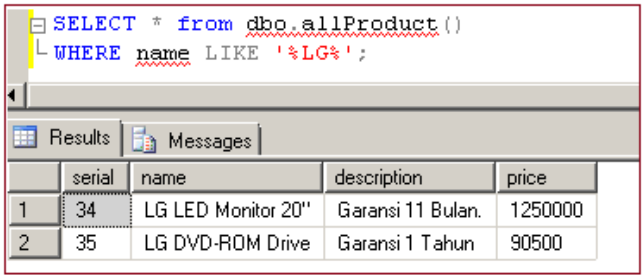
```
CREATE FUNCTION <Inline_Function_Name, sysname,
FunctionName>
(
	-- Add the parameters for the function here
	<@param1, sysname, @p1> <Data_Type_For_Param1, ,
int>,
	<@param2, sysname, @p2> <Data_Type_For_Param2, ,
char>
)
RETURNS TABLE
AS
RETURN
(
	-- Add the SELECT statement with parameter references
here
	SELECT 0
)
```

Pada contoh kali ini akan dibuat ITVF untuk menampilkan seluruh data yang ada pada tabel tertentu tetapi hanya kolom-kolom tertentu.



Gambar 5.43: Fungsi "**allProduct**"

Fungsi **allProduct()** di atas digunakan untuk menampilkan seluruh data yang ada di kolom **serial**, **name**, **description**, dan **price** yang ada di tabel **dbo.products**. sebagaimana aturan tentang ITVF variabel *return* hanya berupa tipe data **TABLE**.



	serial	name	description	price
1	34	LG LED Monitor 20"	Garansi 11 Bulan.	1250000
2	35	LG DVD-ROM Drive	Garansi 1 Tahun	90500

Gambar 5.44: Proses dan hasil eksekusi fungsi "**allProduct**"

Perintah SQL di atas merupakan contoh eksekusi ITVF yang telah dibuat, yaitu fungsi **allProdutc()**. Karena pemanggilan fungsinya ditambahkan klausa **WHERE name LIKE '%LG%'** maka hasil yang dikembalikan berupa data produk yang mengandung kata "**LG**".

5.7.2 Scalar-Valued Function

Scalar-Valued function (disingkat SVF) bisa dibuat dengan atau tanpa parameter. SVF dapat digunakan untuk mengoperasikan kolom-kolom suatu tabel atau *view*. Operasi yang dimaksud dapat berupa operasi matematika, penggabungan *string*, pencarian, dan operasi sejenis lainnya. Adapun sintaks umum dari SVF adalah sebagai berikut.

```
CREATE FUNCTION <Scalar_Function_Name, sysname,
FunctionName>
(
	-- Add the parameters for the function here
	<@Param1, sysname, @p1> <Data_Type_For_Param1, , int>
)
RETURNS <Function_Data_Type, ,int>
AS
BEGIN
	-- Declare the return variable here
	DECLARE <@ResultVar, sysname, @Result>
<Function_Data_Type, ,int>
	-- Add the T-SQL statements to compute the return
value here
	SELECT <@ResultVar, sysname, @Result> = <@Param1,
sysname, @p1>
	-- Return the result of the function
	RETURN <@ResultVar, sysname, @Result>

END
```

Pada contoh kali ini akan dibuat SVF untuk mendapatkan nilai pada kolom (*field*) baru yang dihasilkan dari perkalian dua kolom. Dua kolom ini dapat ditentukan secara langsung saat eksekusi SVF.

```
CREATE FUNCTION subtotal(@qty int, @price float)
RETURNS float
AS
BEGIN
DECLARE @subtot float
SET @subtot = @qty * @price
RETURN @subtot
END
```

Messages

```
Command(s) completed successfully.
```

Gambar 5.45: Fungsi **subtotal()**

Fungsi **subtotal()** di atas digunakan untuk menampilkan hasil kali dua kolom yang di-*parsing* melalui parameter **@qty** dan **@price**. Fungsi ini mengembalikan nilai bertipe data *float*.

```
SELECT orderid, name, quantity, price,
       dbo.subtotal(quantity, price) as SubTotal
FROM dbo.vdetail_order
```

Results | Messages

	orderid	name	quantity	price	SubTotal
1	8	Samsung Galaxy Tab 3	1	5250000	5250000
2	8	Flassdisk 8 GB Kingston	2	79900	159800
3	8	LG LED Monitor 20''	1	1250000	1250000

Gambar 5.46: Proses dan hasil eksekusi fungsi **subtotal()**

Perintah SQL di atas merupakan contoh eksekusi SVF yang telah dibuat, yaitu fungsi **subtotal()**. Karena pemanggilan fungsinya

subtotal(quantity, price) maka hasil yang dikembalikan berupa hasil perkalian dari kolom **quantity** dan **price**.

5.7.3 Multi-Statement Table-Valued Function

Secara umum *multi-statement table-valued function* (disingkat MTVF) mirip dengan *inline table-valued function* (ITVF), perbedaanya hanya pada poin-poin berikut ini.

- Variabel **table** yang akan dikembalikan oleh fungsi harus didefinisikan di awal fungsi.
- MTVF harus mempunyai blok **BEGIN/END**.
- Di dalam blok **BEGIN/END** harus ada kode yang menghasilkan variabel **table**.
- MTVF harus mengembalikan nilai.
- ITVF hanya mengembalikan statemen **SELECT**, tidak ada variabel **table** di dalamnya, tidak ada statemen **INSERT**, dan tidak ada blok kode. Skrip berikut ini merupakan sintaks umum MTVF.

```
CREATE FUNCTION <Table_Function_Name, sysname, FunctionName>
(
	-- Add the parameters for the function here
	<@param1, sysname, @p1> <data_type_for_param1, ,
int>,
	<@param2, sysname, @p2> <data_type_for_param2, ,
char>
)
RETURNS
<@Table_Variable_Name, sysname, @Table_Var> TABLE
(
	-- Add the column definitions for the TABLE variable
here
	<Column_1, sysname, c1> <Data_Type_For_Column1, ,
int>,
	<Column_2, sysname, c2> <Data_Type_For_Column2, ,
int>
)
AS
BEGIN
	-- Fill the table variable with the rows for your
result set
	RETURN
END
```

Untuk contoh MTVF akan digunakan contoh yang telah dibahas pada ITVF dengan sedikit modifikasi.

```
CREATE FUNCTION insCrossTable()
RETURNS @newTable TABLE(id int, nama nvarchar(80), deskripsi nvarchar(MAX), harga float)
AS
BEGIN
  INSERT INTO @newTable (id, nama, deskripsi, harga)
  SELECT serial, name, description, price
    FROM dbo.products
   ORDER BY serial;
  RETURN;
END
```

Messages

```
Command(s) completed successfully.
```

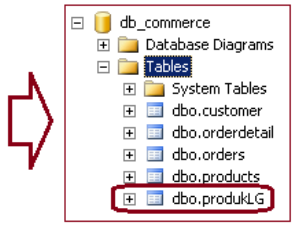
Gambar 5.47: Fungsi **insCrossTable()**

Fungsi **insCrossTable()** di atas digunakan untuk menampilkan data dari tabel **dbo.products** untuk selanjutnya data hasil **SELECT** tersebut disisipkan (*insert*) ke dalam tabel baru.

```
SELECT * INTO produkLG
FROM dbo.insCrossTable() WHERE nama LIKE '%LG%';

SELECT * FROM produkLG;
```

Results | Messages

	id	nama	deskripsi	harga
1	34	LG LED Monitor 20"	Garansi 11 Bulan.	1250000
2	35	LG DVD-ROM Drive	Garansi 1 Tahun	90500



Gambar 5.48: Proses dan hasil eksekusi fungsi **insCrossTable()**

Perintah SQL di atas merupakan contoh eksekusi MTVF yang telah dibuat, yaitu fungsi **insCrossTable()**. Karena pemanggilan fungsinya ditambahkan klausa **WHERE nama LIKE '%LG%'** maka hasil yang dikembalikan berupa data produk yang mengandung kata "**LG**". Data yang dikembalikan oleh fungsi **insCrossTable()** ini selanjutnya disisipkan ke dalam tabel baru dengan nama **produkLG**. Gambar 5.48 (kanan) menunjukkan terciptanya tabel baru dengan nama **dbo.produkLG** hasil eksekusi fungsi **insCrossTable()**.

5.8 Membuat Trigger

Trigger merupakan sebuah bentuk khusus dari *stored procedure* yang secara otomatis melakukan eksekusi ketika sebuah *event* terjadi di

dalam server *database*. *Trigger* dapat mencegah akses terhadap data yang spesifik, melakukan *logging* atau melakukan audit perubahan data. Berikut adalah beberapa manfaat penggunaan *trigger*:

- Sebagai salah satu mekanisme alternatif untuk menjalankan aturan bisnis dan menjaga integritas data dalam sebuah *database*. Mekanisme yang lainnya adalah penggunaan *constraint*.
- Dapat mengevaluasi status dari sebuah tabel sebelum dan sesudah terjadi perubahan data dan melakukan aksinya sesuai perbedaan yang terjadi.
- Dapat melakukan perubahan secara *cascade* melalui relasi tabel pada *database*, akan tetapi perubahan ini akan lebih efisien jika dieksekusi menggunakan *referential integrity constraints*.
- Dapat melakukan pencegahan akses atas data yang kompleks dibanding dengan menggunakan *check constraint*.
- Dapat mereferensikan kolom (*field*) dalam tabel yang berbeda.

Pada SQL Server 2008 R2, *trigger* dikelompokkan menjadi tiga, yaitu: ***Trigger* DML** (*data manipulation language*), ***Trigger* DDL** (*data definition language*), dan ***Trigger* LOGON**.

5.8.1 Trigger DML

Trigger DML melakukan eksekusi ketika ada *user* mencoba melakukan modifikasi data yang behubungan dengan DML. Kejadian (*event*) DML dalam hal ini adalah statemen **INSERT, UPDATE**, dan **DELETE** data pada sebuah tabel atau *view*. Namun demikian, *trigger* tidak dapat digunakan dalam kejadian yang dihasilkan oleh statemen **SELECT**. *Trigger-trigger* ini aktif ketika ada satu lebih *valid event* diaktifkan, terlepas dari ada tidaknya baris data dalam tabel yang terpengaruh (*affected*). Berikut ini merupakan sintaks umum pembuatan *trigger* DML.

```
Trigger on an INSERT, UPDATE, or DELETE statement to a table
or view (DML Trigger)
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ NOT FOR REPLICATION ]
```

```
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME <method
specifier [ ; ] > }
<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
<method_specifier> ::=
    assembly_name.class_name.method_name
```

Tabel 5.8: Keterangan sintaks *Trigger* DML

Argumen	Deskripsi
[schema_name.]trigger_name	Mendeskripsikan nama skema dan *trigger* DML.
{table|view}	Nama tabel atau *view* dimana *trigger* DML akan diterapkan.
<dml_trigger_option>[,...n]	Menentukan opsi *trigger* DML, termasuk klausa ENCRYPTION dan EXECUTE AS.
{[INSERT][,][UPDATE][,][DELETE]}	Opsi *event* untuk *trigger* DML. Untuk menentukan pada *event* apakah (bisa lebih dari satu) *trigger* DML dieksekusi.
[NOT FOR REPLICATION]	Menentukan apakah *trigger* akan dieksekusi saat proses replikasi pada tabel yang bersangkutan terjadi.
{sql_statement[;][,...n] | EXTERNAL NAME <method specifier [;]>}	Perintah-perintah SQL yang akan dieksekusi saat terjadi operasi DML.

5.8.1.1 Trigger untuk Logging

Contoh pertama *trigger* DML kali ini adalah membuat pencatatan (*logging*) terhadap aktifitas tabel "**dbo.products**". Hasil pencatatan ini nantinya disimpan ke dalam tabel "**dbo.log_products**" dengan struktur sebagai berikut.

GENKJAVA-PC.d....log_products

Column Name	Data Type	Allow Nulls
serial	bigint	☐
productid	int	☑
event	nvarchar(50)	☑
date	datetime	☑

Gambar 5.49: Struktur tabel "**dbo.log_products**"

Selanjutnya, buat *trigger* DML dengan nama "**logInsProduct**" dengan kode seperti berikut.

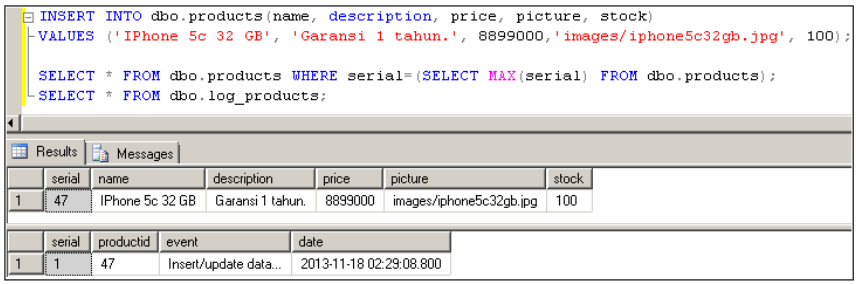
```
CREATE TRIGGER [logInsProducts] ON dbo.products
FOR INSERT, UPDATE
AS
DECLARE
@kode int

SELECT @kode=serial FROM INSERTED;

BEGIN
INSERT INTO dbo.log_products (productid, event, date)
VALUES (@kode, 'Insert/update data...', GETDATE())
END
```

Messages

```
Command(s) completed successfully.
```

Gambar 5.50: Proses pembuatan *trigger* "**logInsProducts**"

Trigger "**logInsProducts**" ini akan terpicu (menjalankan aksi) jika pada tabel "**dbo.products**" dikenai kejadian *insert* atau *update*. Untuk menguji apakah *trigger* "**logInsProducts**" berjalan sebagaimana mestinya, akan dibuat perintah SQL untuk menyisipkan (*insert*) data ke dalam tabel "**dbo.products**" seperti berikut.



```
INSERT INTO dbo.products(name, description, price, picture, stock)
VALUES ('IPhone 5c 32 GB', 'Garansi 1 tahun.', 8899000,'images/iphone5c32gb.jpg', 100);

SELECT * FROM dbo.products WHERE serial=(SELECT MAX(serial) FROM dbo.products);
SELECT * FROM dbo.log_products;
```

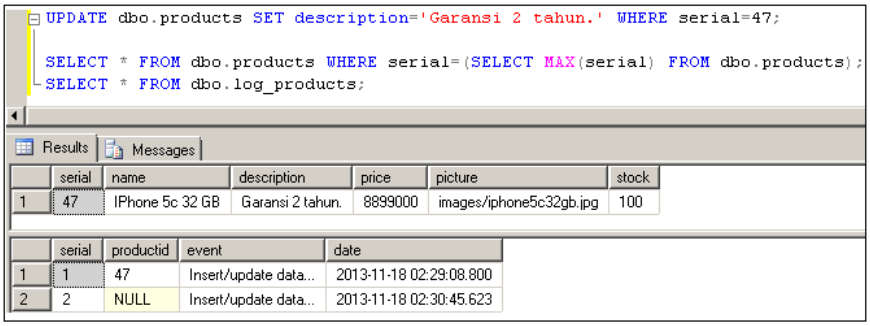
Results | Messages

	serial	name	description	price	picture	stock
1	47	IPhone 5c 32 GB	Garansi 1 tahun.	8899000	images/iphone5c32gb.jpg	100

	serial	productid	event	date
1	1	47	Insert/update data...	2013-11-18 02:29:08.800

Gambar 5.51: Pengujian *trigger* "**logInsProducts**" untuk kejadian *insert*

Gambar 5.51 menunjukkan bahwa *trigger* "**logInsProducts**" terpicu setelah kejadian *insert* dikenakan pada tabel "**dbo.products**". Bukti *trigger* "**logInsProducts**" telah terpicu adalah berhasil disisipkannya sebuah *record* ke dalam tabel "**dbo.log_products**". Selanjutnya, *trigger* "**logInsProducts**" akan diuji dengan kejadian *update* dengan memberikan perintah SQL berikut ini.



```
UPDATE dbo.products SET description='Garansi 2 tahun.' WHERE serial=47;

SELECT * FROM dbo.products WHERE serial=(SELECT MAX(serial) FROM dbo.products);
SELECT * FROM dbo.log_products;
```

Results | Messages

	serial	name	description	price	picture	stock
1	47	IPhone 5c 32 GB	Garansi 2 tahun.	8899000	images/iphone5c32gb.jpg	100

	serial	productid	event	date
1	1	47	Insert/update data...	2013-11-18 02:29:08.800
2	2	NULL	Insert/update data...	2013-11-18 02:30:45.623

Gambar 5.52: Pengujian *trigger* "**logInsProducts**" untuk kejadian *update*

Gambar 5.52 menunjukkan bahwa *trigger* "**logInsProducts**" juga terpicu setelah kejadian *update* dikenakan pada tabel "**dbo.products**". Bukti *trigger* "**logInsProducts**" telah terpicu adalah berhasil disisipkannya lagi sebuah *record* dengan keterangan "**Insert/update data…**" ke dalam tabel "**dbo.log_products**".

Setelah membuat *trigger* DML untuk kejadian *insert* dan *update*, sekarang akan diberikan contoh pembuatan *trigger* untuk kejadian *delete* dengan nama "**logDelProducts**" seperti berikut ini.

```
CREATE TRIGGER [logDelProducts] ON dbo.products
FOR DELETE
AS
DECLARE
@kode int

SELECT @kode=serial FROM DELETED;

BEGIN
INSERT INTO dbo.log_products (productid, event, date)
VALUES (@kode, 'Delete data...', GETDATE())
END
```

Messages

```
Command(s) completed successfully.
```

Gambar 5.53: Proses pembuatan *trigger* "**logDelProducts**"

Trigger "**logDelProducts**" ini akan terpicu jika pada tabel "**dbo.products**" dikenai kejadian *delete*. Untuk menguji apakah *trigger* "**logInsProducts**" berjalan sebagaimana mestinya, akan dibuat perintah SQL untuk menghapus (*delete*) data dari dalam tabel "**dbo.products**" seperti berikut.

```
DELETE FROM dbo.products WHERE serial=47;

SELECT * FROM dbo.products WHERE serial=47;
SELECT * FROM dbo.log_products;
```

Results | Messages

	serial	name	description	price	picture	stock

	serial	productid	event	date
1	1	47	Insert/update data...	2013-11-18 02:29:08.800
2	2	NULL	Insert/update data...	2013-11-18 02:30:45.623
3	3	47	Delete data...	2013-11-18 02:52:27.590

Gambar 5.54: Pengujian *trigger* "**logDelProducts**" untuk kejadian *delete*

Gambar 5.54 menunjukkan bahwa *trigger* "**logDelProducts**" juga terpicu setelah kejadian *delete* dikenakan pada tabel "**dbo.products**". Bukti *trigger* "**logDelProducts**" telah terpicu adalah berhasil disisipkannya lagi sebuah *record* dengan keterangan "**Delete data…**" ke dalam tabel "**dbo.log_products**".

5.8.1.2 Trigger untuk Perbaruan Data

Selanjutnya akan diberikan contoh *trigger* DML dengan operasi yang sedikit lebih kompleks dari sekadar *logging*, yaitu perbaruan data antar tabel yang terrelasi. Contoh *trigger* kali ini diberi nama "**tambahStokProduk**" pada kejadian *insert* data.

```
CREATE TRIGGER tambahStokProduk on dbo.supply
FOR INSERT
AS
UPDATE b SET b.stock = b.stock + a.quantity, b.price = a.price + (a.price/10)
FROM dbo.products b JOIN inserted a ON b.serial = a.productid
```

Messages

```
Command(s) completed successfully.
```

Gambar 5.55: Pembuatan *trigger* "**tambahStokProduk**"

Trigger "**tambahStokProduk**" ini dipicu saat ada proses sisip (*insert*) data ke dalam tabel "**dbo.supply**". Adapun aksi yang dijalankan oleh *trigger* ini adalah memperbarui data "**stock**" dan "**price**" yang ada di tabel "**dbo.products**". Data "**stock**" akan diperbarui dengan nilai stok lama ditambah kuantitas pembelian baru (kolom "**dbo.supply.qty**"), sedangkan data "**price**" diperbarui dengan nilai harga pembelian baru ditambah 10%-nya. Untuk lebih jelasnya, perhatikan proses sisip data ke dalam tabel "**dbo.supply**" berikut ini.

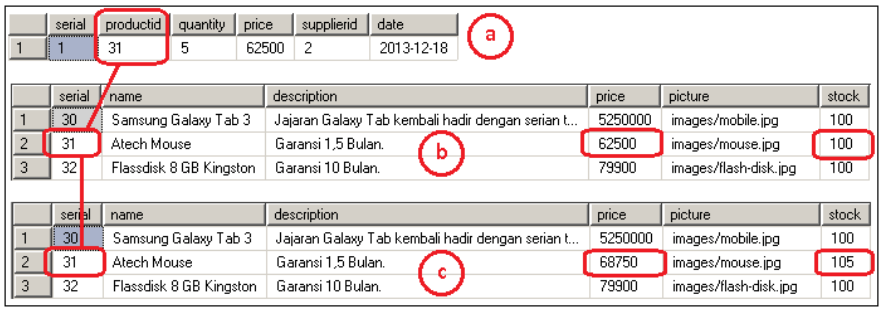
```
INSERT INTO dbo.supply(productid, quantity, price, supplierid, date)
VALUES (31, 5, 62500, 2, GETDATE());

SELECT * FROM dbo.supply;
```

Results | Messages

	serial	productid	quantity	price	supplierid	date
1	1	31	5	62500	2	2013-12-18

Gambar 5.56: Proses sisip data ke dalam tabel "**dbo.supply**"

Sekarang, perhatikan perubahan data yang terjadi pada tabel "**dbo.products**" berikut ini.



	serial	productid	quantity	price	supplierid	date
1	1	31	5	62500	2	2013-12-18

	serial	name	description	price	picture	stock
1	30	Samsung Galaxy Tab 3	Jajaran Galaxy Tab kembali hadir dengan serian t...	5250000	images/mobile.jpg	100
2	31	Atech Mouse	Garansi 1,5 Bulan.	62500	images/mouse.jpg	100
3	32	Flassdisk 8 GB Kingston	Garansi 10 Bulan.	79900	images/flash-disk.jpg	100

	serial	name	description	price	picture	stock
1	30	Samsung Galaxy Tab 3	Jajaran Galaxy Tab kembali hadir dengan serian t...	5250000	images/mobile.jpg	100
2	31	Atech Mouse	Garansi 1,5 Bulan.	68750	images/mouse.jpg	105
3	32	Flassdisk 8 GB Kingston	Garansi 10 Bulan.	79900	images/flash-disk.jpg	100

Gambar 5.57 Hasil eksekusi *trigger* "**tambahStokProduk**"
(a) Tabel "**dbo.supply**" **(b)** Tabel "**dbo.products**" sebelum perubahan;
(c) Tabel "**dbo.products**" setelah perubahan.

Perhatikan perubahan yang terjadi pada tabel "**dbo.products**" di atas. Untuk data produk dengan **ID=31** (Astech Mouse), data stok (*stock*) menjadi **105** (seratus lima) dan data harga (*price*) menjadi **68750** (enam puluh delapan ribu tujuh ratus lima puluh). Data stok **105** didapatkan dari **100** (stok lama) ditambah dengan **5** (kuantitas pembelian baru), sedangkan **68750** didapat dari **62500** (harga baru) ditambah dengan **6250** (atau **10%** dari harga baru tersebut). Berikut ini disajikan contoh terakhir untuk *trigger* DML, yaitu "**ubahStokProduk**".

```
CREATE TRIGGER ubahStokProduk on dbo.supply
FOR DELETE
AS
UPDATE b SET b.stock = b.stock - a.quantity
FROM dbo.products b JOIN DELETED a ON b.serial = a.productid
```

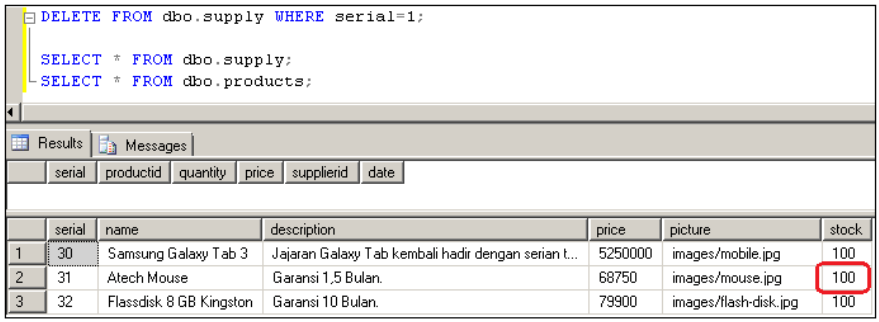
Messages

```
Command(s) completed successfully.
```

Gambar 5.58: Pembuatan *trigger* "**ubahStokProduk**"

Trigger "**ubahStokProduk**" ini akan dipicu saat terjadi proses penghapusan (*deleting*) data produk tertentu pada tabel "**dbo.supply**". Aksi yang dilakukan *trigger* ini adalah mengurangi stok yang ada di dalam tabel "**dbo.products**" sebesar kuantitas data produk yang dihapus

pada tabel "**dbo.supply**". Untuk lebih jelasnya, perhatikan proses hapus data dari dalam tabel "**dbo.supply**" berikut ini.



```
DELETE FROM dbo.supply WHERE serial=1;

SELECT * FROM dbo.supply;
SELECT * FROM dbo.products;
```

Results | Messages

serial	productid	quantity	price	supplierid	date

	serial	name	description	price	picture	stock
1	30	Samsung Galaxy Tab 3	Jajaran Galaxy Tab kembali hadir dengan serian t...	5250000	images/mobile.jpg	100
2	31	Atech Mouse	Garansi 1,5 Bulan.	68750	images/mouse.jpg	100
3	32	Flassdisk 8 GB Kingston	Garansi 10 Bulan.	79900	images/flash-disk.jpg	100

Gambar 5.59: Proses hapus data pada tabel "**dbo.supply**"

Perhatikan perubahan yang terjadi pada tabel "**dbo.products**" di atas. Untuk data produk dengan **ID=31** (Astech Mouse), data stok (*stock*) menjadi **100** (seratus lima). Data stok **100** didapatkan dari **105** (stok lama) dikurangi dengan **5** (kuantitas data pada tabel "**dbo.supply**").

5.8.2 Trigger DDL

Trigger DDL melakukan eksekusi dalam menanggapi berbagai *event* DDL. *Event-event* yang tergolong DDL adalah statemen **CREATE**, **ALTER**, dan **DROP**, termasuk *stored procedure* bawaan sistem yang melakukan operasi sejenis DDL. Berikut ini merupakan sintaks umum pembuatan *trigger* DDL.

```
Trigger on a CREATE, ALTER, DROP, GRANT, DENY, REVOKE, or
UPDATE STATISTICS statement (DDL Trigger)
CREATE TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH <ddl_trigger_option> [ ,...n ] ]
{ FOR | AFTER } { event_type | event_group } [ ,...n ]
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME < method
specifier >  [ ; ] }

<ddl_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

Di bawah ini diberikan contoh pembuatan *trigger* DDL dengan "**log**" yang akan digunakan untuk melakukan pencatatan (*log*) terhadap

seluruh kejadian DDL (DDL *event*), seperti pembuatan (*create*), perbaruan struktur (*alter*) dan penghapusan (*drop*) seluruh tabel yang ada di dalam *database* "**db_commerce**". Pertama, dibuat sebuah tabel "**ddl_log**" untuk menyimpan pencatatan yang dihasilkan *trigger* "**log**". Selanjutnya, perhatikan kode SQL pembuatan *trigger* "**log**" berikut ini.

```
USE db_commerce;
GO
CREATE TABLE ddl_log (PostTime datetime, DB_User nvarchar(100),
                      Event nvarchar(100), TSQL nvarchar(2000));
GO
CREATE TRIGGER log
ON DATABASE
FOR DDL_DATABASE_LEVEL_EVENTS
AS
DECLARE @data XML
SET @data = EVENTDATA()
INSERT ddl_log
   (PostTime, DB_User, Event, TSQL)
   VALUES
   (GETDATE(),
   CONVERT(nvarchar(100), CURRENT_USER),
   @data.value('(/EVENT_INSTANCE/EventType)[1]', 'nvarchar(100)'),
   @data.value('(/EVENT_INSTANCE/TSQLCommand)[1]', 'nvarchar(2000)') );
GO
```

Messages

```
Command(s) completed successfully.
```

Gambar 5.60: Proses pembuatan *trigger* "**log**"

Untuk melihat hasil pencatatan yang dihasilkan *trigger* "**log**", berikut diberikan contoh kode SQL untuk membuat (*create*) dan menghapus (*drop*) sebuah tabel dengan nama "**TestTable**" di dalam *database* "**db_commerce**".

```
CREATE TABLE TestTable (a int)
DROP TABLE TestTable ;
GO
SELECT * FROM ddl_log ;
GO
```

Results | Messages

	PostTime	DB_User	Event	TSQL
1	2013-12-18 22:41:06.183	dbo	CREATE_TABLE	CREATE TABLE TestTable (a int)
2	2013-12-18 22:41:06.187	dbo	DROP_TABLE	DROP TABLE TestTable ;

Gambar 5.61: Proses pengujian (*testing*) *trigger* "**log**"

Pada Gambar 5.61 di atas menunjukkan bahwa seluruh *event* DDL yang terjadi pada seluruh komponen *database* "**db_commerce**", baik berupa tabel atau *view*, maka *trigger* "**log**" akan terpicu dan melakukan pencatatan sesuai *event* yang terjadi.

5.8.3 Trigger LOGON

Trigger LOGON melakukan eksekusi *stored procedure* (bawaan sistem) dalam menanggapi *event* LOGON yang dibangkitkan ketika sebuah sesi pengguna (*user session*) berhasil dibentuk dengan sebuah instan dari SQL Server. *Trigger* LOGON aktif setelah fase otentikasi login dalam tahap akhir, tetapi sebelum sesi pengguna benar-benar terbentuk. Oleh karena itu, semua pesan yang dihasilkan oleh *trigger* yang biasanya sampai ke pengguna, seperti pesan *error* dan pesan dari statemen **PRINT**, dialihkan ke catatan kesalahan (*error log*) SQL Server. *Trigger* LOGON tidak aktif jika otentikasi gagal.

Anda dapat menggunakan *trigger* LOGON untuk mengaudit dan mengontrol sesi server, seperti penulusuran aktifitas login, membatasi login ke SQL Server, atau pembatasan jumlah sesi untuk login tertentu. Berikut ini merupakan sintaks umum pembuatan *trigger* LOGON.

```
Trigger on a LOGON event (Logon Trigger)
CREATE TRIGGER trigger_name
ON ALL SERVER
[ WITH <logon_trigger_option> [ ,...n ] ]
{ FOR| AFTER } LOGON
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME < method
specifier >  [ ; ] }

<logon_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

Untuk contoh kali ini akan dibuat sebuah *trigger* untuk membatasi waktu login *user* tertentu. Batasan yang dimaksud adalah jam login, misalkan *user* tertentu, anggap saja "**TestUser**", bisa login ke SQL Server hanya pada pukul **10:00 WIB** sampai dengan **18:00 WIB**.

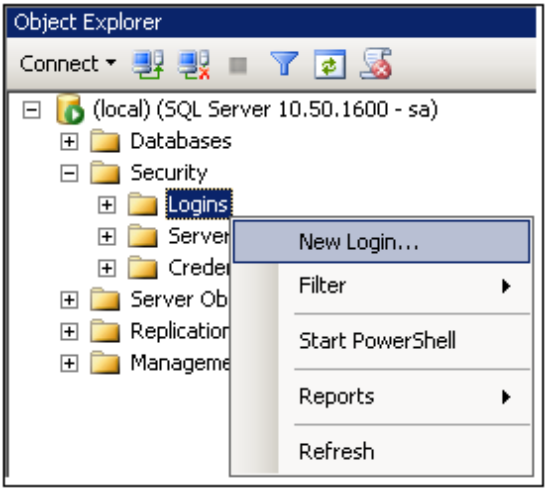
Pertama kali yang perlu dibuat adalah membuat login (*user*) baru yang akan dibatasi jam loginnya. Ada 2 cara untuk membuat login baru, yaitu: dengan dengan mengetikkan baris perintah SQL langsung melalui *query editor* yang tersedia atau menggunakan *wizard* melalui

aplikasi GUI. Berikut adalah perintah SQL untuk membuat login baru dengan nama "**TestUser**".

```
USE [master]
GO
--Create the login on your servel called "TestUser"
CREATE LOGIN [TestUser] WITH PASSWORD=N'Admin2345'
    ,DEFAULT_DATABASE=[master]
    ,DEFAULT_LANGUAGE=[us_english]
    ,CHECK_EXPIRATION=OFF
    ,CHECK_POLICY=OFF
GO
```

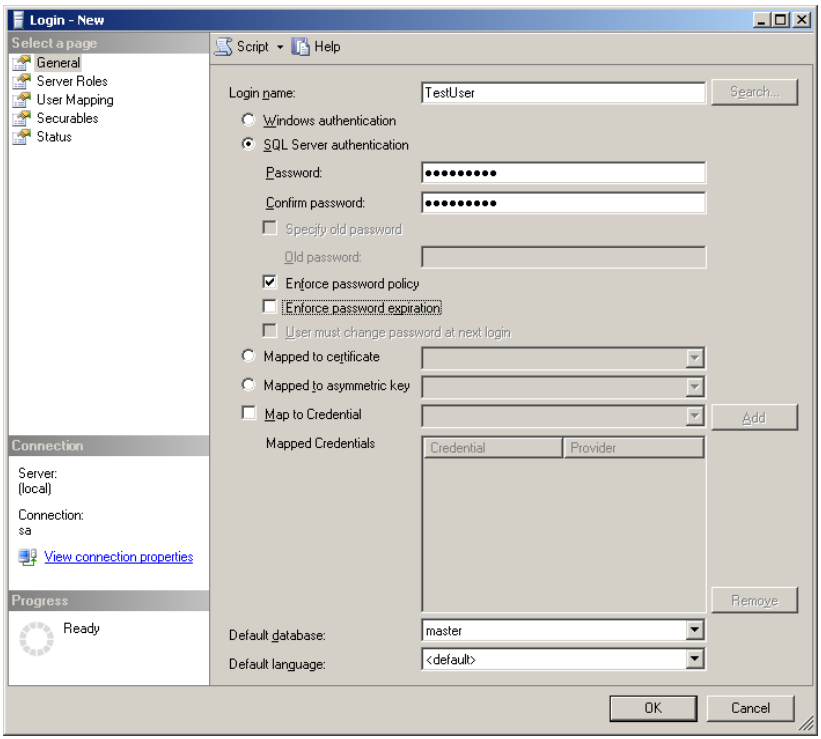
Adapun dengan *wizard* melalui aplikasi GUI langkah-langkahnya adalah sebagai berikut.

1) Setelah berhasil masuk ke SQL Server, klik kanan submenu **Security » Logins**, pilih **New Login...**.



Gambar 5.62: Proses membuat login baru

2) Setelah muncul jendela **Login – New**, ketik "**TestUser**" pada kolom **Login name**.

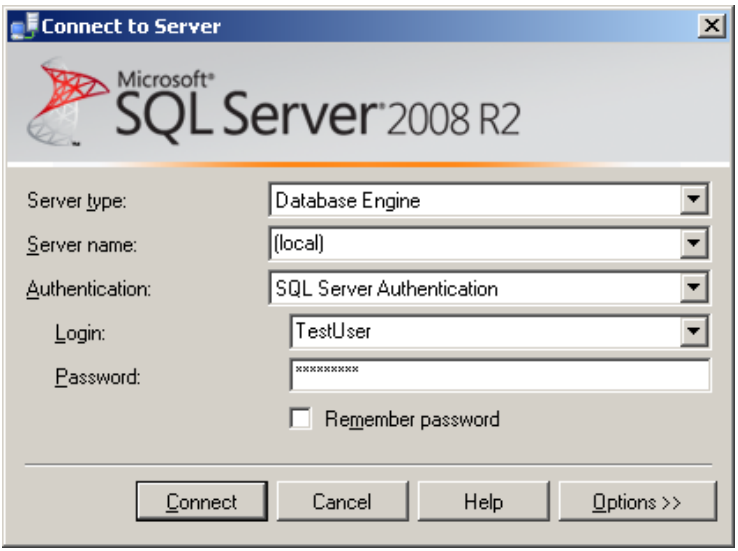


Gambar 5.63: Jendela **Login - New**

3) Pilih opsi **SQL Server authentication** dan tentukan kata sandinya melalui kolom **Password** dan **Confirm password**, misalkan "**Admin2345**" (tanpa tanda kutip).
4) Hilangkan tanda (*uncheck*) pada opsi **Enforce password expiration**.
5) Tentukan nama *default database* untuk *user* "**TestUser**" melalui kolom **Default database**, misalkan "**master**".
6) Klik tombol **[OK]**, jika tidak muncul pesan kesalahan, berarti proses pembuatan login baru berhasil.

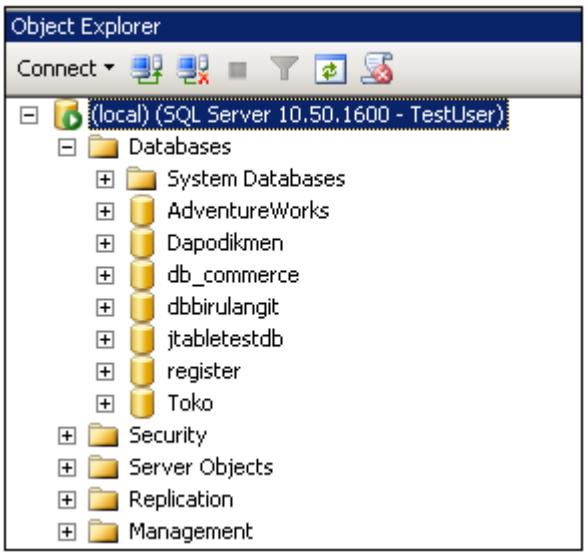
Langkah selanjutnya adalah memutuskan koneksi dari SQL Server dan mencoba login lagi menggunakan *user* baru yang telah dibuat, yaitu: "**TestUser**".

1) Klik tombol  (*disconnect*) untuk memutuskan koneksi dengan SQL Server.
2) Setelah muncul jendela **Connect to Server**, pilih nama server melalui kolom **Server name**.
3) Pilih opsi **SQL Server Authentication** pada kolom **Authentication**.
4) Pilih **TestUser** pada kolom **Login** dan masukkan *password* "**Admin2345**" sebagaimana yang telah dibuat sebelumnya.



Gambar 5.64: Jendela **Connect to Server**

5) Mestinya Anda akan berhasil masuk ke SQL Server dan dibawa ke halaman utama SSMS dengan bagian **Object Explorer** menampilkan daftar *database* yang ada di dalamnya seperti Gambar 5.65 berikut ini.



Gambar 5.65: Bagian **Object Explorer** pada halaman utama **SSMS**

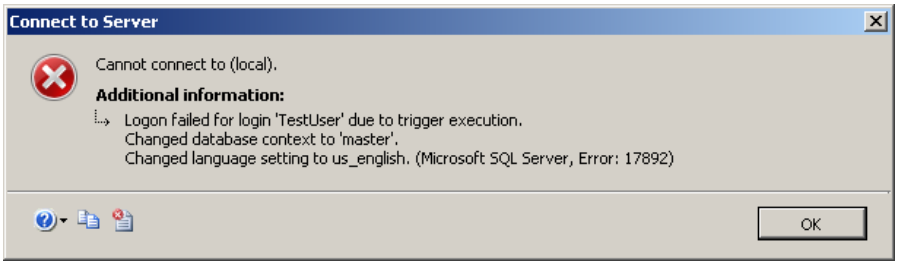
Sekarang saatnya membuat *trigger* untuk membatasi jam koneksi, misalkan diberi nama "**connection_limit_trigger**", dengan kode SQL seperti berikut ini.

```
USE [master]
GO

CREATE TRIGGER [connection_limit_trigger]
ON ALL SERVER
FOR LOGON
AS
BEGIN
DECLARE @ErrorText [varchar](128)

SET @ErrorText = 'Tidak diijinkan login dengan "TestUser" di luar jam kerja. '
SET @ErrorText = @ErrorText + 'Silakan coba lagi pada jam 10:00 s.d 18:00 WIB.'

IF ORIGINAL_LOGIN() = 'TestUser' AND
    (DATEPART(HOUR, GETDATE()) < 10 OR DATEPART (HOUR, GETDATE()) > 18)
    BEGIN
        PRINT @ErrorText
        ROLLBACK;
    END
END;
GO

ENABLE TRIGGER [connection_limit_trigger] ON ALL SERVER
GO
```

```
Messages
Command(s) completed successfully.
```

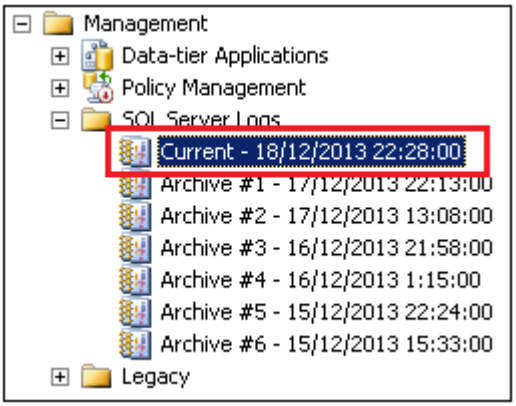
Gambar 5.66: Proses pembuatan *trigger* "**connection_limit_trigger**"

Untuk mengujinya, putuskan koneksi SQL Server dan coba lagi login menggunakan "**TestUser**" sebagaimana yang diilustrasikan pada Gambar 5.64. Jika Anda melakukan login tidak di antara pukul **10:00WIB** dan **18:00WIB**, maka akan muncul pesan kesalahan seperti berikut ini.



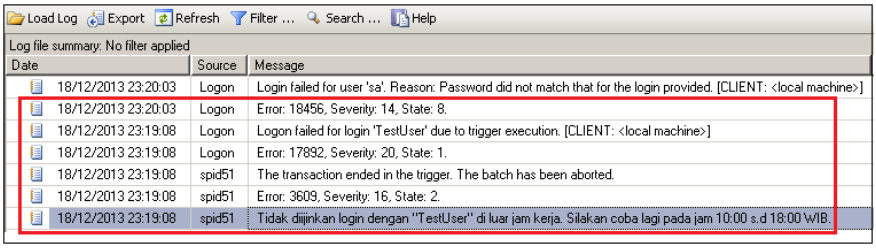
Gambar 5.67: Pesan kesalahan yang dihasilkan oleh *trigger* "**connection_limit_trigger**"

Selanjutnya, Anda coba lagi login menggunakan *user* lain yang tidak dibatasi jam loginnya, misal "**sa**". Setelah berhasil masuk ke SQL Server, coba akses menu **Management** yang ada pada **Object Explorer**, kemudian klik catatan (*log*) terakhir yang ada dalam menu **SQL Server Log**.



Gambar 5.68: Submenu **SQL Server Log** pada **Object Explorer**

Gambar 5.69 berikut ini menunjukkan catatan yang dihasilkan oleh *trigger* "**connection_limit_trigger**".



Gambar 5.69: Catatan yang dihasilkan oleh *trigger* "**connection_limit_trigger**"

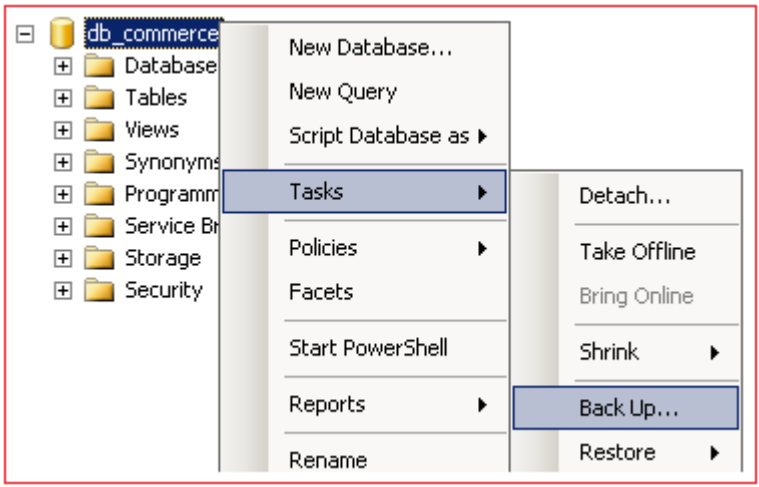
Perhatikan baris paling bawah pada Gambar 5.67, pesan yang ditampilkan adalah "Tidak diijinkan login dengan "TestUser" di luar jam kerja...", pesan ini sebagaimana yang telah didefinisikan saat pembuatan *trigger* "**connection_limit_trigger**".

Kesimpulan yang dapat diambil tentang *trigger* Logon adalah *trigger* Logon sangat berguna untuk melacak dan mengendalikan aktivitas login SQL Server. Informasi yang ditangkap oleh *trigger* Logon membantu kita mengidentifikasi dan mencegah akses yang tidak sah dari SQL Server.

5.9 Backup Database

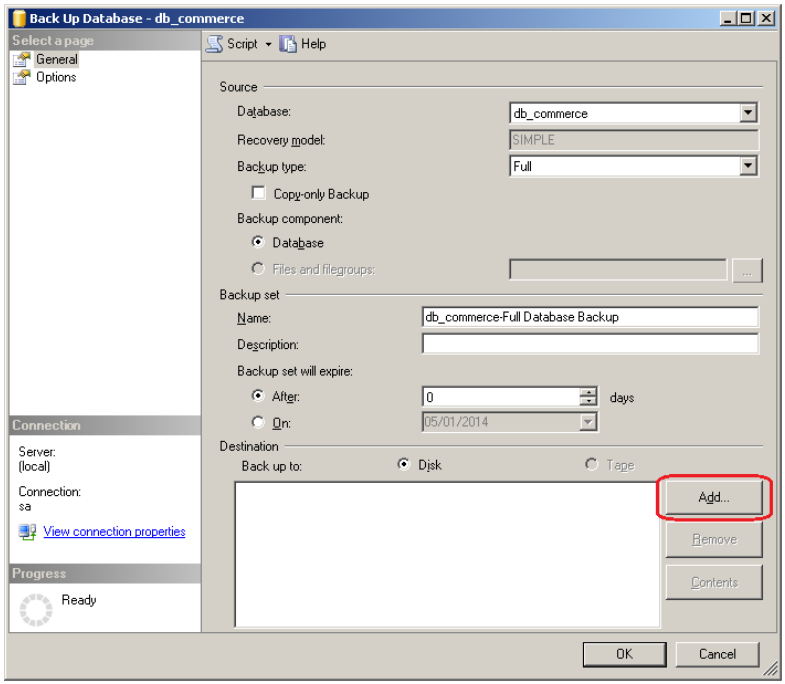
Biasanya *backup database* dilakukan sebagai tindakan antisipatif agar aman dari kerusakan atau kehilangan data (termasuk fisik servernya), baik yang diakibatkan oleh manusia (seperti: *cracking*, pencurian, dan tindakan sabotase) maupun alam (seperti: banjir, kebakaran, dan bencana alam lainnya). *Backup database* dapat dilakukan secara langsung atau periodik. Berikut ini adalah langkah-langkah untuk melakukan *backup database* melalui SSMS.

1) Pada bagian **Object Explorer**, klik kanan nama *database* yang akan di-*backup*, kemudian akses menu **Tasks » Back Up...**.



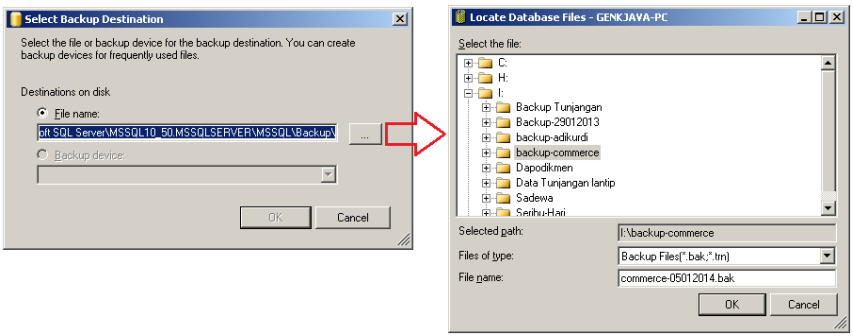
Gambar 5.70: Menu **Tasks » Back Up...**

2) Setelah muncul jendela **Back Up Database**, tentukan nama *database* yang akan di-*backup*, misal "**db_commerce**" pada kolom **Database** dan pada kolom **Backup type**, pilih opsi "**Full**" jika diinginkan *backup* keseluruhan.



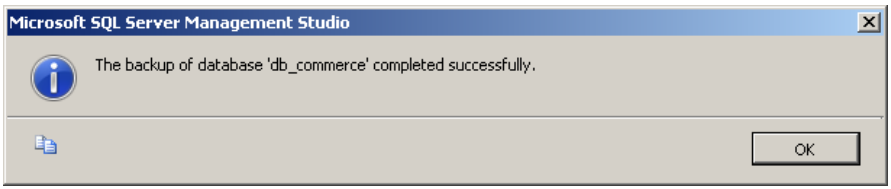
Gambar 5.71: Jendela **Back Up Database**

3) Klik tombol **[Add...]** untuk menentukan nama file tujuan (hasil *backup*) dan alamat (*path*) file hasil *backup* tersebut akan disimpan.
4) Setelah muncul jendela **Select Backup Destination**, klik tombol **[...]** (baca: **browse**) sampai muncul jendela **Locate Database Files**, tentukan direktori penyimpanan (misal: "**I:\backup-commerce**") dan nama file hasil *backup* yang diinginkan (misal: "**commerce-05012014.bak**"), klik tombol **[OK]**.



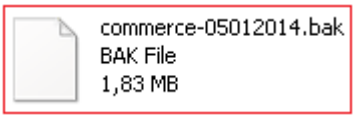
Gambar 5.72: Jendela **Select Back Up Destination** dan **Locate Database Files**

5) Setelah kembali ke jendela **Select Backup Destination** dan **Back Up Database** (Gambar 5.71) klik tombol **[OK]**. Jika berhasil, maka akan muncul pesan sukses seperti Gambar 5.73.



Gambar 5.73: Pesan sukses *backup database*

6) Gambar 5.74 berikut ini merupakan file hasil *backup database* yang berekstensi ***.bak**.

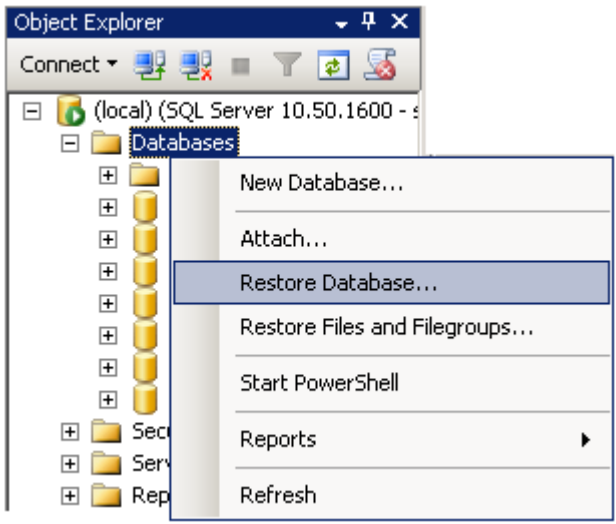


Gambar 5.74: File hasil *backup database*

5.10 Restore Database

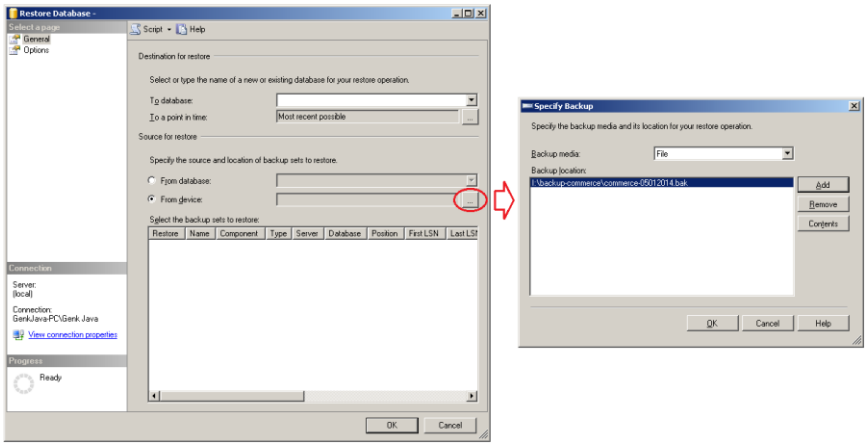
Restore database merupakan tindakan mengembalikan seluruh komponen *database* melalui file hasil proses *backup*, tujuannya agar konten *database* sama dengan data yang ada di dalam file hasil *backup*. Berikut adalah langkah melakukan *restore database* melalui SSMS.

1) Pada bagian **Object Explorer**, klik kanan nama *database* yang akan di-*backup*, kemudian akses menu **Restore Database...**.



Gambar 5.75: Menu **Restore Database**

2) Setelah muncul jendela **Restore Database**, pada bagian **Source to restore** pilih opsi **From device** dan klik tombol telusur (tiga titik).

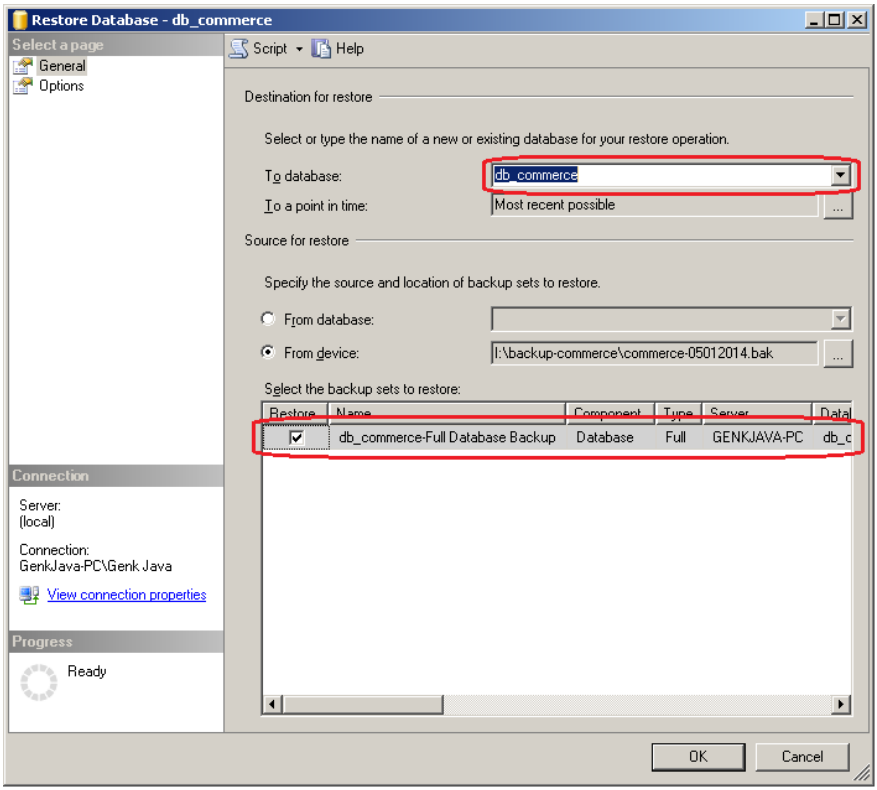


Gambar 5.76: Jendela **Restore Database**

3) Pada jendela **Specify Backup**, klik tombol **Add**. Setelah muncul jendela **Locate Backup File**, tentukan file *backup* yang akan di-

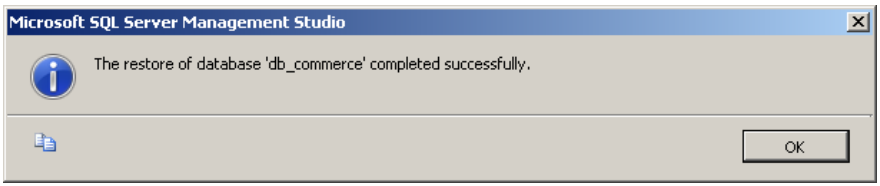
restore, klik tombol **[OK]**. Setelah kembali ke jendela **Specify Backup**, klik tombol **[OK]**.

4) Pada jendela **Restore Database**, pada bagian **Destination to restore**, pilih "**db_commerce**" yang muncul pada kolom **To database**.



Gambar 5.77: Bagian **Destination for restore** pada jendela **Restore Database**

5) Jangan lupa beri tanda centang file *backup* pada bagian **Select the backup sets to restore**, klik tombol **[OK]**. Jika berhasil, maka akan muncul pesan sukses seperti Gambar 5.78.

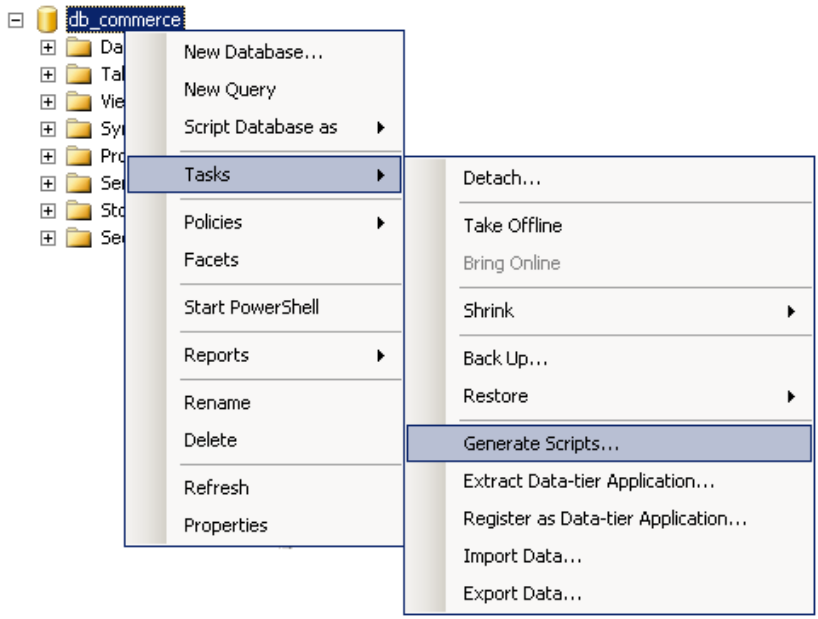


Gambar 5.78: Pesan sukses *restore database*

5.11 Generate Skrip

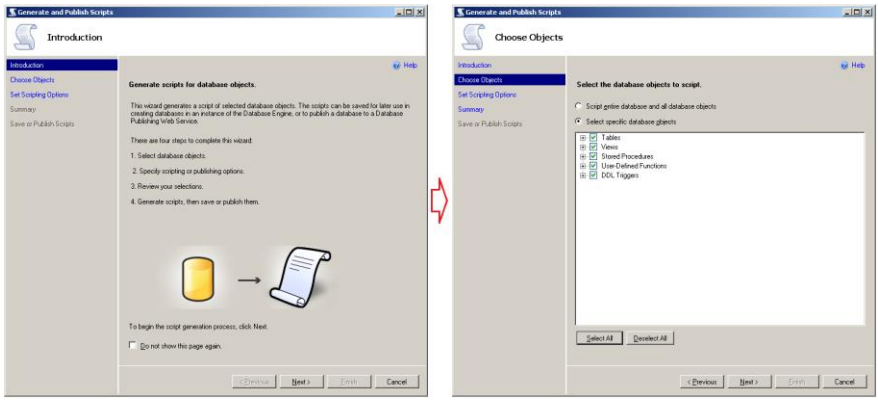
Generate skrip dapat dikatakan sebagai proses *backup* dalam format *.SQL, sehingga file hasil *generate* ini mudah dibaca dan dipahami isinya karena berupa kumpulan perintah-perintah SQL. Keuntungan lain dari proses ini adalah dapat melakukan *backup* berdasarkan komponen *database* tertentu, misalkan tabel, *trigger*, atau *stored procedure* tertentu saja. Untuk lebih jelasnya, perhatikan langkah-langkahnya berikut ini.

1) Pada bagian **Object Explorer**, klik kanan nama *database* yang akan di-*generate*, kemudian akses menu **Tasks » Generate Scripts...**.



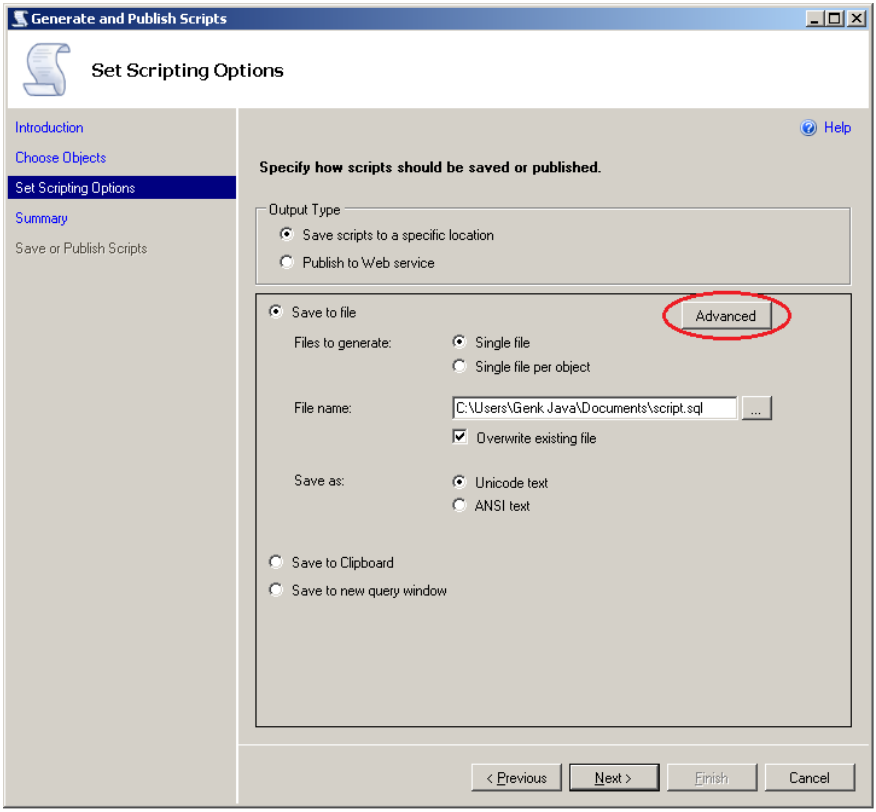
Gambar 5.79: Proses *generate* skrip

2) Setelah muncul jendela **Generate and Publish Scripts (**disingkat **GPS) - Introduction**, klik tombol **[Next]**.
3) Pada jendela **GPS – Choose Objects**, pilih objek *database* mana saja yang ingin di-*generate* skripnya, kemudian klik tombol **[Next]**.



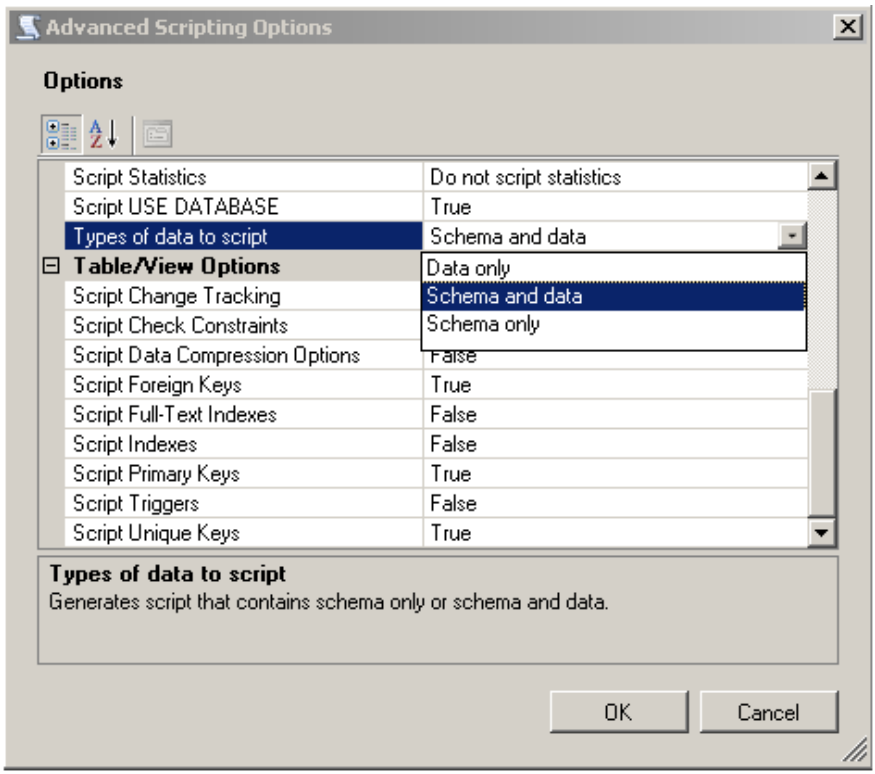
Gambar 5.80: Jendela **GPS Introduction** dan **GPS Choose Objects**

4) Pada jendela **GPS – Set Scripting Options**, pilih opsi **Save to file**. Klik tombol **[Advanced]** untuk melakukan pengaturan tingkat lanjut.



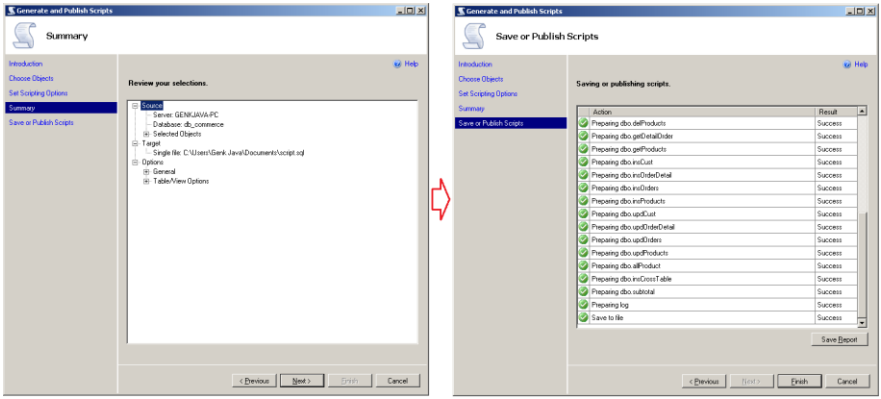
Gambar 5.81: Jendela **GPS Set Scripting Options**

5) Setelah muncul jendela **Advanced Scripting Options**, lakukan pengaturan yang dianggap perlu, misalkan pengaturan untuk opsi **Type of data to script** diset dengan **Schema and data**. Jika selesai, klik tombol **[OK]**.



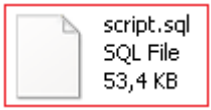
Gambar 5.82: Jendela **Advanced Scripting Options**

6) Pada jendela **GPS - Summary**, langsung klik saja tombol **[Next]**, sampai muncul jendela **GPS – Save or Publish Scripts** klik tombol **[Finish]**.



Gambar 5.83: Jendela **GPS – Summary** dan **GPS – Save and Publish Scripts**

7) Jika berhasil, maka file "**scrrip.sql**" akan terbentuk di dalam direktori yang ditentukan.



Gambar 5.84: File hasil *generate* skrip

Demikian sedikit tentang administrasi *database* Ms. SQL Server, Anda dapat mencari informasi dari internet atau media lain untuk membantu Anda memperdalam bahasan materi pada bab ini.

Daftar Pustaka

Riyanto. 2014. Membuat Sendiri Aplikasi Web Store dengan PHP, jQuery & Microsoft SQL Server. Gava media

Sumber Internet:

1. http://blogs.msdn.com, [Diakses: 01 September 2014, 01:30]
2. http://stackoverflow.com, [Diakses: 05 Oktober 2014, 02:15]
3. http://technet.microsoft.com, [Diakses: 06 September 2014, 03:00]
4. http://www.microsoft.com/en-us/download/details.aspx?id=20098, [Diakses: 22 September 2014, 00:11]
5. http://social.msdn.microsoft.com, [Diakses: 17 Oktober 2014, 02:12]
6. http://sqlcrudphpwizard.codeplex.com, [Diakses: 19 Oktober 2014, 01:45]

Profil Penulis

Dr. Lantip Diat Prasojo. Lahir di Magetan, 25 April 1974. Saat ini tercatat sebagai dosen tetap di Prodi Manajemen Pendidikan Fakultas Ilmu Pendidikan dan Pascasarjana UNY. Penulis menyelesaikan S1 Teknik Elektro di UGM, S2 Manajemen Pendidikan di UNY (memperoleh gelar Magister Pendidikan dalam waktu 19 bulan dengan predikat *Cumlaude*) dan S3 Prodi Administrasi/Manajemen Pendidikan UPI Bandung (memperoleh gelar Doktor dalam waktu 2 tahun dengan predikat *Cumlaude*). Beberapa mata kuliah yang diampu adalah Manajemen Strategik, TQM, Praktik Manajemen Pendidikan, Manajemen Pendidikan, Sistem Informasi Manajemen (SIM), TIK Manajemen dan Manajemen Perkantoran. Penulis pernah ditugasi UNY sebagai Ketua Laboratorium Jurusan Administrasi Pendidikan, Koordinator ISO Pascasarjana UNY, Manajer LIMUNY Puskom UNY, Sekretaris Eksekutif Rektor UNY, dan Sekretaris Prodi S2 dan S3 Manajemen Pendidikan Pascasarjana UNY. Selain itu, penulis juga pernah membantu Dirjen PMPTK Kementerian Pendidikan Nasional dalam Proyek BERMUTU, Fasilitator tingkat nasional untuk diklat kepala sekolah, Narasumber Nasional di P2TK Dikmen, Tim CPD (*Continuous Professional Development*) untuk Kepala Sekolah di seluruh Indonesia dan sebagai Asesor BAN PT Kemdiknas. Penulis dapat dihubungi melalui email: **lantip1975@gmail.com**.

Lantip Diat Prasojo

PERANCANGAN DATABASE SISTEM INFORMASI MANAJEMEN PENDIDIKAN

dengan DBMS Microsoft (Acces dan SQL Server)

Secara teknis, dalam buku ini di level *database* Anda akan dituntun untuk membuat **Database SIM Perpustakaan dengan Microsoft Access**, Penggunaan SQL melalui Fitur Query, Penerapan SQL pada Satu Tabel, Penerapan Nested SQL (Sub-Query), Penerapan SQL pada Multi Tabel, **Membuat Form dan Laporan dengan Microsoft Access, Pendahuluan, Membuat Form, Mengubah Text Box menjadi Combo Box, Mengubah Text Box Menjadi Option Group, Membuat Report. Selain itu juga dipandu terkait Instalasi Microsoft SQL Server dan Konfigurasinya,** Kebutuhan Sistem, Instalasi SQL Server 2008 R2, a**dministrasi Database Microsoft SQL Server,** Membuat Database, Membuat Tabel, Membuat Diagram Database, Persiapan Data, Membuat View, Membuat Stored Procedure, Membuat UDF, Membuat Trigger. Backup Database, Restore Database, Generate Skrip.

ISBN 602-7981-35-0

UNY PRESS

Jl.H.Affandi (Jl.Gejayan), Gg. Alamanda,
Kompleks FT-UNY, Kampus Karangmalang, Yogyakarta,
Kode Pos: 55281,Telp.(0274)589346,
unypress.yogyakarta@gmail.com