BUKU AJAR

Pengantar Bahasa Query

Oleh: Arsito Ari Kuncoro, M.Kom

PROGRAM STUDI S1-TEKNIK INFORMATIKA
UNIVERSITAS SAINS DAN TEKNOLOGI KOMPUTER
UNIVERSITAS STEKOM
2021

Pengantar Bahasa Query

Oleh: Arsito Ari Kuncoro, M.Kom

YAYASAN PRIMA AGUS TEKNIK

Pengantar Bahasa Query

Penulis:
Arsito Ari Kuncoro, S.Kom., M.Kom

ISBN : 21238-2812817-22

Editor:
Danang., S.Kom., M.Kom

Penyunting :
Hendri Rasminto, S.Kom.,M.Si

Desain Sampul dan Tata Letak :
Sulartopo ,S.Kom., M.Kom

Penerbit :
Yayasan Prima Agus Teknik
Redaksi:
Jln Majapahit No 605 Semarang
Tlpn. (024) 6723456
Fax . 024-6710144
Email: penerbit_ypat@stekom.ac.id

Distributor Tunggal:
UNIVERSITAS STEKOM
Jln Majapahit No 605 Semarang
Tlpn. (024) 6723456
Fax . 024-6710144
Email: info@stekom.ac.id

KATA PENGANTAR

Kehandalan dari suatu sistem database atau DBMS seperti Microsoft SQL Server, Oracle, PostgreSQL, dan sejenisnya dapat diketahui dari cara kerja optimizer-nya dalam memproses statement-statement SQL yang dibuat oleh user maupun program-program aplikasinya. Di dalam optimizer, statement-statement yang ada diproses dengan salah satu cara dari banyak cara yang ada untuk mendapatkan perencanaan query yang paling optimal sehingga pada akhirnya akan didapatkan jawaban query dengan waktu akses yang paling minimum.

Diktat ini disusun dan memudahkan mahasiswa – mahasiswa STEKOM memahami dasar - dasar pengolahan datas menggunakan perintah Query dengan database Micsosoft SQL Server sebagai praktikumnya.

Harapan penyusun, buku ini bisa bermanfaat dan bisa digunakan oleh mahasiswa dan pembaca pada umumnya.

Agustus 2021

Penyusun

DAFTAR ISI

DAFTAR PUSTAKA

Petroutsos, Evangelos, *Mastering Database Programming with Visual Basi, Sybex Inc,2000*

Gunderloy,Mike, *Visual Basic Developer's Guide to ADO and SQL Server7 in Record Time*, Endicott, Washington,, 1999

SQL Server, *Programmer Guide*

SQL Server, *SQL Server Query Analyzer Help*

BAB I
PENDAHULUAN

1.1 DBMS

Dewasa ini, komputer mempunyai peranan penting dalam segala segi kehidupan. Kebanyakan dari komputer-komputer ini digunakan untuk menyimpan dan mendapatkan kembali data yang mempunyai jumlah yang besar dalam sebuah cara yang efisien, seperti sistem-sistem yang biasa disebut dengan *database system* dan software yang mengatur jalannya data (keluar masuknya data) yang biasa disebut dengan *database management system (DBMS)*. Fasilitas dari DBMS adalah dapat mengakses sebuah database tunggal secara bersamaan oleh banyak user, dapat mengakses data secara terbatas hanya untuk user yang berhak dan mengganti kegagalan dari sistem-sistem tanpa kehilangan keutuhan data. Umumnya, interface yang pertama (pokok) untuk sebuah DMBS adalah sebuah high level query atau *data manipulation language* yang dapat dengan mudah digunakan. Contoh dari *high level query* adalah SQL (*Structure Query Language*).

Statement – statement (pernyataan-pernyataan) dalam SQL dapat dihasilkan secara langsung oleh user dengan menggunakan command-command (perintah-perintah) interface atau dengan sebuah program aplikasi.

Kehandalan dari suatu sistem database atau DBMS dapat diketahui dari cara kerja optimizer-nya dalam memproses statement-statement SQL yang dibuat oleh user maupun program-program aplikasinya. Di dalam optimizer, statement-statement yang ada diproses dengan salah satu cara dari banyak cara yang ada untuk mendapatkan perencanaan query yang paling optimal sehingga pada akhirnya akan didapatkan jawaban query dengan waktu akses yang paling minimum. Proses untuk mencari perencanaan eksekusi query yang terbaik inilah yang disebut dengan *proses optimisasi query*.

1.2 PENGENALAN PROSES dan OPTIMATISI QUERY

Database Manajemen Sistem (DBMS) adalah kumpulan dari program-program yang membolehkan user untuk menciptakan dan memelihara sebuah database. DBMS sudah menjadi peralatan standar untuk melindungi pengguna komputer dari bagian-bagian kecil dalam pengelolaan *secondary storage (hard disk)*. DBMS didesain untuk meningkatkan produktivitas dari aplikasi para programmer dan untuk memberikan kemudahan pengaksesan data oleh komputer.

Pengaksesan data di dalam DBMS dapat dilakukan dengan berbagai macam cara. Dan tentunya dalam melakukan pengaksesan data ada hal-hal yang perlu diperhatikan seperti ketepatgunaan implementasi dari data itu sendiri serta waktu prosesnya. Ada banyak *plan* (rencana) yang dapat diikuti oleh database manajemen sistem

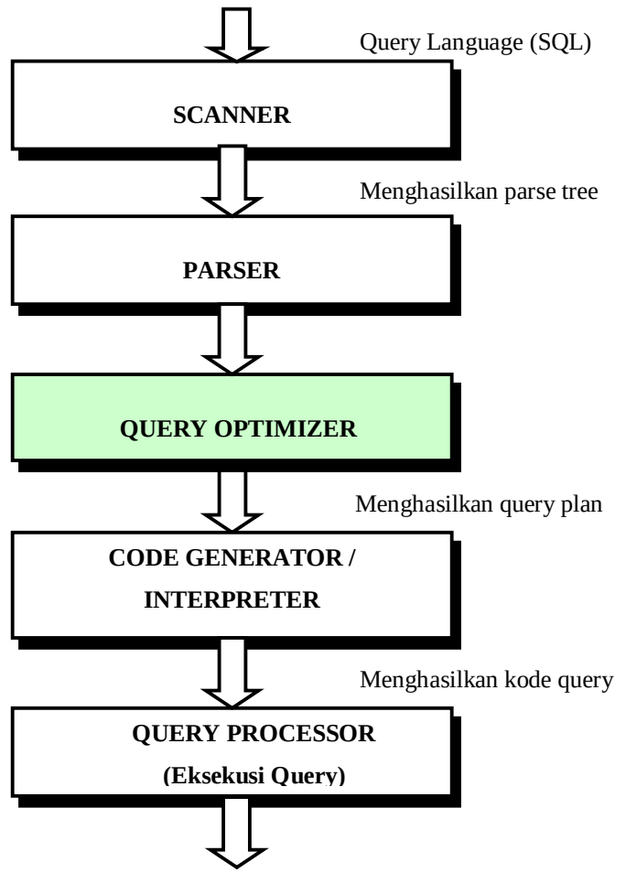
dalam memproses dan menghasilkan jawaban sebuah query. Semua plan pada akhirnya akan menghasilkan jawaban (output) yang sama tetapi pasti mempunyai harga yang berbeda-beda, seperti misalnya total waktu yang diperlukan untuk menjalankan sebuah query.

Optimisasi query mencoba memberikan suatu pemecahan untuk menangani masalah tersebut dengan cara menggabungkan sejumlah besar teknik-teknik dan strategi, yang meliputi transformasi-transformasi logika dari query-query untuk mengoptimisasi jalan akses dan penyimpanan data pada sistem file. yang diminta.

Sebuah query yang diekspresikan dalam sebuah bahasa query tingkat tinggi seperti SQL mula-mula harus dibaca, diuraikan dan disahkan (*scanning*, *parsing*, *validating*)[1]. Query tersebut kemudian dibentuk menjadi sebuah struktur data yang biasa disebut dengan *query tree*. Dan kemudian DBMS (Database Manajemen Sistem) harus merencanakan sebuah strategi eksekusi untuk mendapatkan kembali hasil dari query dari file-file database. Tahapan-tahapan proses dari sebuah query di dalam sebuah sistem database ditunjukkan pada gambar 1.1. Berikut penjelasan dari masing-masing tahapan :

- *Scanner* melakukan identifikasi (pengenalan) token-token seperti *SQL keywords*, *attribute*, dan *relation name*. Proses ini disebut dengan *scanning*.

[1] http://cisnet.baruch.cuny.edu/holowczak/classes/9440/queryprocessing/



Gambar 1.1. Tahapan Proses QUERY

- *Query Parser* mengecek kevalidan query dan kemudian menterjemahkannya ke dalam sebuah bentuk internal yaitu ekspresi relasi aljabar atau parse tree. Proses ini disebut dengan *parsing*.
- *Query Optimizer* memeriksa semua ekspresi-ekspresi aljabar yang sama untuk query yang diberikan dan memilih salah satu dari ekspresi tersebut yang terbaik yang memiliki perkiraan termurah. Dengan kata lain, tugas dari query optimizer adalah menghasilkan sebuah rencana eksekusi. Proses ini disebut dengan *optimisasi query*.
- *Code Generator* atau *Interpreter* mentransformasikan rencana akses yang dihasilkan oleh optimizer ke dalam kode-kode. Setelah itu, kode-kode tersebut dikirimkan ke dalam query processor untuk dijalankan.
- *Query Processor* melakukan eksekusi query untuk mendapatkan hasil query yang diinginkan.

Bagian yang diarsir pada gambar 1.1 adalah merupakan komponen utama yang berperan penting dalam proses optimisasi query.

Tahapan proses query pada gambar 1.1 memberikan gambaran yang jelas tentang bagaimana pada umumnya sebuah query diproses di dalam sebuah Database Manajemen Sistem (DBMS).

1.3 Pengenalan Tentang Query

Sebuah *query* adalah sebuah ekspresi bahasa yang menggambarkan data yang akan didapatkan kembali dari sebuah database. Dalam hubungannya dengan optimisasi query, seringkali diasumsikan bahwa query-query tersebut dinyatakan dalam sebuah dasar-dasar isi dan sekumpulan cara orientasi, yang memberikan optimizer pilihan-pilihan diantara alternatif prosedur-prosedur evaluasi.

Query dapat digunakan pada beberapa keadaan. Kebanyakan aplikasi nyatanya adalah permintaan-permintaan secara langsung dari *user* yang memerlukan informasi tentang bentuk maupun isi dari database. Apabila permintaan user terbatas pada sekumpulan query-query standar, maka query-query tersebut dapat dioptimisasi secara manual oleh pemrograman prosedur-prosedur pencarian gabungan dan membatasi input dari user pada sebuah ukuran menu. Tetapi bagaimanapun juga, sebuah sistem optimisasi query otomatis menjadi penting apabila query-query khusus ditanyakan dengan menggunakan bahasa query yang digunakan secara umum seperti SQL.

Aplikasi yang kedua dari query terjadi pada transaksi-transaksi yang mengubah data yang disimpan berdasarkan nilainya saat itu. Pada akhirnya, query seperti ekspresi-ekspresi dapat digunakan secara internal dalam sebuah DBMS, sebagai contoh

adalah untuk mengecek kebenaran akses dan menyamakan kebenaran akses-akses yang terjadi.

Membicarakan tentang query, sangat erat hubungannya dengan cara penulisan query tersebut ke dalam sebuah bentuk bahasa yang mudah dimengerti. Pada umumnya, bahasa query yang digunakan untuk mengekspresikan sebuah pernyataan dari query adalah SQL (Structure Query Language).

SQL adalah sebuah bahasa database yang luas yang memiliki *statement-statement* (pernyataan) untuk definisi data, query dan *update* data (memperbaharui data). SQL mempunyai satu statement dasar untuk mendapatkan kembali informasi dari sebuah database. Statement dasar dari SQL adalah SELECT.

Bentuk dasar dari statement SELECT biasa disebut dengan blok *select from where* yang terbentuk dari tiga macam klausa yaitu SELECT, FROM dan WHERE yang mempunyai bentuk sebagai berikut :

SELECT<daftar Attribute>
FROM <daftar Tabel>
WHERE <kondisi>

Dimana <*daftar attribute*> adalah sebuah daftar dari nama-nama attribute yang nilai-nilainya didapatkan oleh query. Sedangkan <*daftar tabel*> adalah sebuah daftar dari nama-nama relasi yang

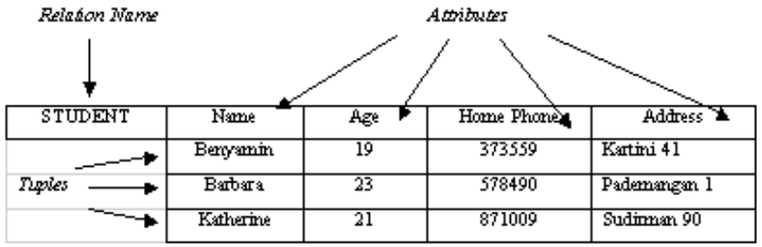
diperlukan oleh proses sebuah query. <*kondisi*> adalah sebuah kondisi ekspresi boolean yang mengidentifikasikan tuple-tuple yang akan dikembalikan oleh query.

Selanjutnya, statement *Select-From-Where* akan selalu digunakan dalam pembahasan mengenai proses dan teknik optimisasi query nanti dengan tujuan untuk lebih memudahkan pemahaman tentang proses optimasi query karena statement tersebut merupakan statement dasar yang mudah dimengerti dan umum digunakan.

1.4 Hubungan Database Dengan Pemrosesan Query

Database adalah kumpulan dari data-data yang berhubungan satu sama lainnya yang digunakan untuk pencarian suatu data tertentu pada saat SQL query dijalankan. Sebuah database dirancang, dibuat dan ditempati oleh data dengan tujuan tertentu. Di dalam sistem database relasional, tabel-tabel dari database saling berhubungan satu sama lainnya. Dan sebuah tabel database akan selalu memiliki *attribute names* (nama-nama attribute), *relation names* (nama-nama relasi), dan *tuples* (record-record).

Untuk lebih jelasnya, pada gambar 1.2 diberikan sebuah tabel database yang lengkap dengan attribute, relation name dan tuples.



Gambar 1.2

Attributes dan tuples dari relasi STUDENT

Attribute digunakan untuk mengidentifikasikan sebuah nama yang diikutsertakan dalam relasi dan menspesifikasikan domain (tipe data sederhana yang menentukan sebuah pemisahan data). Sedangkan *tuples* adalah kumpulan dari record-record di dalam sebuah database. *Relation name* mendefinisikan attribute-attribute yang diperlukan dalam predikat dan mendefinisikan arti dari predikat tersebut.

1.5 Sistem Katalog dalam DBMS

Sistem katalog adalah sebuah sistem yang digunakan untuk menyimpan informasi-informasi mengenai suatu database. Informasi yang disimpan dalam sebuah katalog dari sebuah relasional database manajemen sistem meliputi relation names, attribute names, dan attributes domains (tipe-tipe data), gambaran dari batasan-batasan (*primary key*, *secondary key*, *foreign key*, NULL/NOT NULL dan

tipe-tipe dari batasan-batasan lainnya), dan bentuk-bentuk penyimpanan dan index-index. Query optimizer melakukan akses pada katalog untuk jalan akses, implementasi (pelaksanaan) informasi, dan data statistik untuk menentukan jalan terbaik untuk menjalankan sebuah query.

Tabel 1.1

Contoh katalog yang menyimpan informasi tentang database

TABLE_NAME	NUM_ROWS	BLOCK
PROJECT	2000	100
DEPARTMENT	50	5
EMPLOYEE	10000	2000

Sebagai contoh, optimizer mengakses katalog untuk mengecek field-field mana dari sebuah relasi yang memiliki akses hash atau akses index-index, sebelum memutuskan bagaimana untuk menjalankan sebuah kondisi pilihan ataupun kondisi join pada relasi. Contoh dari sebuah katalog yang menyimpan informasi tentang database dapat dilihat pada tabel 1.1

BAB II

DESAIN BASIS DATA

2.1 Desain Basis Data Rasional

Desain yang tidak baik adalah memiliki sifat

1. Pengulangan informasi (information redudancy)
2. Ketidakmampuan merepresentasikan informasi tertentu

Contoh pengulangan Informasi :

Dept	KplDept	Lokasi	Nama	NIP	Ruang
AA	Dedi	Majapahit	Agus	9901	AA-1
AA	Dedi	Majapahit	Joko	9902	AA-3
AA	Dedi	Majapahit	Heru	9905	AA-4
BB	Dono	Hasanudin	Totok	9908	BB-2
BB	Dono	Hasanudin	Dini	9910	BB-3

Contoh ketidak mampuan merepresentasikan informasi tertentu :

- yang punya KTP tapi tidak punya SIM
- yang tidak punya KTP dan SIM

Nama	Sex	KTP	SIM	Umur
Agus	P	10001	A250969	21
Budi	P	20203	C090973	19
Anik	W	40044	C101070	20

2.2 Aturan Normalisasi

Dalam perancangan database ada beberapa aturan, salah satunya yang disebut dengan aturan normalisasi. Aturan ini akan membantu dalam merancang database yang normal atau setidaknya memverifikasi rancangan database.

Definisi Normalisasi Data adalah proses yang berkaitan dengan model data relasional untuk mengorganisir himpunan data dengan ketergantungan dan keterkaitan yang tinggi / erat.

Kegunaan Normalisasi Data :

- Meminimalisasi pengulangan informasi
- Menghilangkan kesulitan pemutakhiran data, misal : insert, modification, delete
- Memudahkan identifikasi entiti / object

Beberapa bentuk Normalisasi Data :

- Bentuk Normal I (First Normal Form – 1NF)
- Bentuk Normal II (Second Normal Form – 2NF)
- Bentuk Normal III (Third Normal Form – 3NF)
- Bentuk Normal IV (Fourth Normal Form – 4NF)
- Bentuk Normal Boyce-Codd (Boyce-Codd Normal Form – BCNF)
- Project Join Normal Form (PJNF)

- Domain Key Normal Form (DKNF)
- Bentuk Normal V (Fifth Normal Form – 5NF)

Database dikatakan normal jika ia tidak mengulangi informasi atau tidak menimbulkan keanehan (anomali) pada proses update atau penghapusan. Aturan atau bentuk normalisasi sendiri ada beberapa macam (bervariasi), namun pada umumnya hanya ada tiga yang sering digunakan, yaitu : aturan normalisasi pertama / First Normal Form (1NF), kedua (2NF) dan ketiga (3NF).

Yang perlu diperhatikan bahwa tabel harus memenuhi 1NF sebelum bisa menerapkan aturan normalisasi ke dua. Yang terakhir, tabel yang terbentuk 2NF bisa dinormalisasi berdasarkan aturan ke tiga.

2.2.1 Perancangan Database Yang Salah

Berikut ini tabel yang menyimpan informasi mengenai buku-buku :

Tabel Judul

ISBN	Judul	Kategori	Hal
979-9550-26-2	Belajar ASP	Internet	120
979-9550-26-2	Belajar ASP	Pemrograman	120

Tabel ini menunjukkan kelemahan, sebuah buku dengan judul “Belajar ASP” bisa dikategorikan dalam topik *Internet*, tetapi juga bisa dimasukkan ke dalam *Pemrograman*. Jadi terdapat pengulangan

data judul buku untuk setiap kategori, demikian juga informasi data lainnya seperti ISBN dan halaman.

Tujuan utama dalam merancang database adalah menghindari duplikasi informasi (*redudancy*), untuk menghindari duplikasi yang tidak diinginkan ini, maka harus memindah kategori ke tabel lain.
Mungkin ada yang mempertimbangkan dengan menambahkan kolom untuk banyak topik, tetapi juga harus menentukan jumlah maksimal kategori dari topik per judul, sehingga tampak seperti berikut :

Tabel Judul

ISBN	Judul	Kategori1	Kategori2	Kategori3
979-9550-26-2	Belajar ASP	Internet	Pemrograman	

Hal
120

Tabel ini bahkan lebih parah, karena untuk menghindari duplikasi data tabel, malah memperkenalkan duplikasi informasi pada struktur tabel itu sendiri. Dalam aturan normalisasi pertama (1NF) syaratnya adalah menghindari kelompok-kelompok kolom dengan informasi yang memiliki jenis yang sama.
Misalnya, *Pemrograman* diganti dengan *Bahasa Pemrograman*, maka harus mengubah terlalu banyak baris pada terlalu banyak tempat, beberapa judul buku mungkin memiliki kategori *pemrograman* di

bawah field Kategori1, yang lain di bawah field Kategori2, dan sebagainya.

2.2.2 Mengatasi dengan Banyak Tabel

Untuk memecahkan masalah banyak kolom dengan tipe sama seperti di atas, maka harus dibuatkan dua tabel baru atau lebih seperti berikut:

Tabel Judul

ISBN	Judul	Hal
979-9550-26-2	Belajar ASP	120

Tabel JudulKategori

ISBN	KodeKategori
979-9550-26-2	Prg
979-9550-26-2	Int

Tabel DaftarKategori

KodeKategori	NamaKategori
Prg	Pemrograman
Int	Internet

Pada rancangan ini :

- Menggunakan sebuah tabel *DaftarKategori* yang menyimpan topik-topik saja, sehingga bila ingin mengubah keterangan dari sebuah topik, cukup mengganti satu baris saja dalam tabel *DaftarKategori*.

- Jika ada judul buku yang harus dikategorikan dibawah banyak topik, maka harus disisipkan banyak baris pada tabel *JudulKategori.*
- Untuk mengetahui topik dari sebuah buku, gunakan ISBN buku tersebut untuk mencari dalam tabel *JudulKategori*, kemudian berdasarkan field *KodeKategori* dari baris-baris yang ditemukan digunakan untuk mencari *NamaKategori* dari judul buku tersebut.

Kelihatannya rumit, tetapi ini lebih sederhana dari kelihatannya, karena semua tabel diindeks dengan tepat, dan DBMS akan mampu mencarikan informasi yang dibutuhkan dengan benar.

2.2.3 Menambahkan Beberapa Informasi dan Menganalisa Kembali

Tambahkan field-field baru pada tabel *Judul*, seperti berikut :

Tabel Judul

ISBN	Judul	Hal	ThnTerbit	Penerbit	AlmPenerbit
979-9550-26-2	Belajar ASP	120	2003	Airlangga	Jl. Erlangga 12

Bila diperhatikan, struktur ini membawa ke dalam situasi yang sangat tidak diinginkan, seperti :

- Jika ada penerbit yang pindah, maka harus mengubah alamat dan bukan hanya satu, tapi bisa ratusan atau ribuan record (*update*

anomaly). Jika ada yang lupa satu baris (record), maka database akan memberi informasi yang buruk.

- Jika menghapus buku terakhir dari sebuah penerbit, maka akan kehilangan semua informasi mengenai penerbit tersebut (*delete anomaly*).

Situasi ini dapat dihindari dengan memindahkan penerbit ke tabel yang berbeda sebagai berikut :

Tabel Judul

ISBN	Judul	Hal	ThnTerbit	KodePenerbit
979-9550-26-2	Belajar ASP	120	2003	Arg

Tabel DaftarPenerbit

KodePenerbit	Penerbit	AlmPenerbit
Arg	Airlangga	Jl. Erlangga 12

Jadi untuk menemukan identitas lengkap penerbit suatu judul buku, maka harus mengambil *KodePenerbit* dari tabel *Judul* lalu mencari baris (record) pad tabel *DaftarPenerbit.*

Field *KodePenerbit* pada tabel *DaftarPenerbit* menjadi kunci primer (*primary key)* karena ia tidak bisa muncul lebih dari satu baris, sedangkan field *KodePenerbit* pada tabel *Judul* adalah kunci asing *(foreign key*) sehingga ia bisa muncul pada banyak baris.

2.2.4 Bentuk Normal Pertama (1NF)

Aturan : Suatu relasi memenuhi 1NF jika dan hanya jika setiap atribut dari relasi tersebut hanya memiliki nilai tunggal dalam satu baris / record atau dengan kata lain sebuah tabel tidak boleh mengandung kelompok yang berulang.

Berikut contoh tabel yang mengandung kelompok yang berulang.

Tabel Judul

ISBN	Judul	Kategori1	Kategori2	Kategor3	Hal
979-9550-26-2	Belajar ASP	Internet	Pemrograman		120

Penerbit	AlamatPenerbit
Airlangga	Jl.Erlangga 12

Untuk menghapus sebuah kelompok dari tabel, biarkan saja kolom pertama dari kelompok tersebut dan ubahlah topik-topik tambahan menjadi baris-baris pada tabel, seperti berikut :

Tabel Judul

ISBN	Judul	Kategori	Hal
979-9550-26-2	Belajar ASP	Internet	120
979-9550-26-2	Belajar ASP	Pemrograman	120

Penerbit	AlamatPenerbit
Airlangga	Jl.Erlangga 12
Airlangga	Jl. Erlangga 12

Bentuk normal pertama tidak membutuhkan tabel yang dipecah-pecah ke dalam banyak tabel. Ia hanya mengubah beberapa kolom menjadi baris-baris tambahan. Dengan ciri khusus :

1. Tidak ada field yang kosong
2. Tidak ada batasan

2.2.5 Bentuk Normal Kedua (2NF)

Aturan :

- Memenuhi 1NF
- Setiap atribut bukan kunci utama bergantung secara fungsional terhadap semua atribut kunci dan bukan hanya sebagian atribut kunci atau dengan kata lain setiap field yang tidak bergantung sepenuhnya pada kunci primer harus dipindahkan ke tabel lain.

Field *kategori* tidak bergantung secara fungsional terhadap ISBN. Bergantung secara fungsional adalah sebuah field sangat ditentukan oleh kunci. Oleh sebab itu pada aturan normal kedua membutuhkan agar topik-topik dipindah ke tabel yang berbeda.

Tabel Judul

ISBN	Judul	Hal
979-9550-26-2	Belajar ASP	120

Penerbit	AlamatPenerbit
Airlangga	Jl. Erlangga 12

Tabel JudulKategori

ISBN	Kategori
979-9550-26-2	Pemrograman
979-9550-26-2	Internet

Struktur tabel ini lebih baik, dimana ISBN yang sama dapat membawa ke banyak topik. Tabel *JudulKategori* tidak mengulang informasi. Hanya kunci primer yang diulang dan ini tidak dapat dihindari.

Tetapi ada masalah lagi, tentang *update anomaly* yaitu jika anda mengubah keterangan dari sebuah kategori, maka harus banyak mengubah banyak baris. Untuk menghindari anomali ini, harus membuat sebuah tabel terpisah yang berisi topik-topik dan menentukan ID yang unik untuk setiap topik.

Tabel Judul

ISBN	Judul	Hal
979-9550-26-2	Belajar ASP	120

Penerbit	AlamatPenerbit
Airlangga	Jl. Erlangga 12

Tabel JudulKategori

ISBN	KodeKategori
979-9550-26-2	Prg
979-9550-26-2	Int

Tabel DaftarKategori

KodeKategori	NamaKategori
Prg	Pemrograman
Int	Internet

2.2.6 Bentuk Normal Ketiga (3NF)

Aturan :

- Memenuhi 2NF
- setiap atribut bukan kunci tidak bergantung secara fungsional kepada atribut bukan kunci yang lain dalam relasi tersebut atau dengan kata lain tidak boleh ada kebergantungan antar field-field non kunci.

Pada struktur tersebut memiliki ketergantungan, yaitu *alamat penerbit* tergantung pada *penerbit* bukan pada pada ISBN buku.

Untuk menghapus ketergantungan ini, harus memindahkan informasi penerbit ke tabel lain.

Tabel Judul

ISBN	Judul	Hal	ThnTerbit	KodePenerbit
979-9550-26-2	Belajar ASP	120	2003	Arg

Tabel DaftarPenerbit

KodePenerbit	Penerbit	AlmPenerbit
Arg	Airlangga	Jl. Erlangga 12

Tabel JudulKategori

ISBN	KodeKategori
979-9550-26-2	Prg
979-9550-26-2	Int

Tabel DaftarKategori

KodeKategori	NamaKategori
Prg	Pemrograman
Int	Internet

Latihan :

Buat normalisasi untuk kasus nilai mahasiswa dengan tabel bentuk belum normal berikut:

Tabel Mahasiswa

NPM NamaMhs Mk1 Mk2 Mk3 Sks1 Sks2 Sks3 Nil1 Nil2 Nil3 Dosen AlmDosen

BAB III
OPTIMASI QUERY

3.1 Optimisasi Query

Optimisasi Query adalah suatu proses untuk menganalisa query untuk menentukan sumber-sumber apa saja yang digunakan oleh query tersebut dan apakah penggunaan dari sumber tersebut dapat dikurangi tanpa merubah output. Atau bisa juga dikatakan bahwa optimisasi query adalah sebuah prosedur untuk meningkatkan strategi evaluasi dari suatu query untuk membuat evaluasi tersebut menjadi lebih efektif. Optimisasi query mencakup beberapa teknik seperti transformasi query ke dalam bentuk logika yang sama, memilih jalan akses yang optimal dan mengoptimumkan penyimpanan data.

Optimisasi query merupakan bagian dasar dari sebuah sistem database dan juga merupakan suatu proses untuk menghasilkan rencana akses yang efisien dari sebuah query di dalam sebuah database. Secara tidak langsung, sebuah rencana akses merupakan sebuah strategi yang nantinya akan dijalankan untuk sebuah query, untuk mendapatkan kembali operasi-operasi yang apabila dijalankan akan menghasilkan database record query. Ada tiga aspek dasar yang ditetapkan dan mempengaruhi optimisasi query, yaitu : *search space, cost model* dan *search strategy*.

Search space adalah sekumpulan rencana-rencana akses yang sama secara logika yang dapat digunakan untuk mengevaluasi sebuah query. Semua rencana-rencana dalam search space query mengembalikan hasil yang sama biarpun beberapa rencana lebih efisien dibandingkan dengan rencana yang lainnya.

Cost model menandakan sebuah harga untuk tiap rencana dalam search space. Harga dari rencana tersebut adalah sebuah perkiraan dari sumber-sumber yang digunakan pada saat rencana dijalankan, dimana harga yang lebih rendah, merupakan yang terbaik dari rencana-rencana yang ada.

Search strategy adalah sebuah perincian dari rencana-rencana mana dalam search space yang akan diperiksa. Apabila search space-nya kecil, maka strategi yang dapat diteruskan adalah menghitung dan mengevaluasi setiap rencana. Meskipun kebanyakan search space bahkan untuk query-query yang sederhana adalah sangat besar, akan tetapi query optimizer selalu memerlukan aturan heuristik untuk mengontrol nomer dari rencana-rencana yang akan diperiksa.

3.2 Sejarah Singkat Optimisasi Query

Optimisasi query lahir sejak sebelum tahun 1970. Pada saat itu yang dikenal dengan jaman kegelapan (dark age), optimisasi query masih dilakukan secara manual oleh manusia. Selain itu, pada jaman tersebut database relasional juga belum dikenal sehingga diperlukan

seseorang yang benar-benar ahli dalam database untuk melakukan optimisasi query. Jadi pada jaman kegelapan hingga tahun 1970-an, optimisasi query masih dilakukan oleh manusia dan masih menggunakan cara yang benar-benar kuno.

Seiring dengan perkembangan jaman di mana teknologi menjadi semakin maju, maka sekitar pertengahan tahun 1970-an sampai pada pertengahan tahun 1980-an optimisasi query tidak lagi dilakukan secara manual, tetapi dilakukan oleh suatu sistem yang dikenal dengan *sistem R* yang kemudian menjadi berkembang pada optimisasi perintah JOIN, sekumpulan tahapan untuk optimisasi query selanjutnya. Hingga saat ini, optimisasi query terus berkembang bersamaan dengan bermunculannya teknik-teknik baru yang digunakan dalam proses optimisasi query meskipun pada dasarnya hanya ada dua macam teknik utama yang biasanya digunakan dalam mengoptimisasi sebuah query. Bisa dikatakan bahwa optimisasi query merupakan tulang punggung dari sebuah sistem karena ketepatgunaan suatu sistem sangat tergantung dari optimizernya.

Hampir semua DBMS menggunakan sistem optimisasi query untuk mengurangi waktu eksekusi query sehingga kerja sistem dapat dioptimalkan dan penggunaan sumber-sumber dapat diminimumkan. Selain itu, akes dari disk yang sangat lambat dibandingkan dengan akses memory membuat optimisasi query menjadi semakin penting.

3.3 Tujuan Optimisasi Query

Prinsip ekonomi yang diperlukan untuk sebuah query adalah mengoptimisasi prosedur-prosedur, mencoba untuk memaksimumkan output dari sejumlah sumber-sumber yang diberikan ataupun untuk meminimumkan penggunaan sumber untuk memberikan output.

Tujuan dari optimisasi query adalah berbeda-beda untuk setiap sistem. Ada yang menggunakan optimisasi query untuk meminimumkan waktu proses sedangkan pada situasi lain bisa juga optimisasi query diperlukan untuk waktu respon, meminimumkan I/O dan meminimumkan penggunaan memory. Tetapi pada dasarnya, tujuan dari optimisasi query adalah menemukan jalan akses yang termurah untuk meminimumkan total waktu pada saat proses sebuah query. Untuk mencapai tujuan tersebut, maka diperlukan *optimizer* untuk melakukan analisa query dan untuk melakukan pencarian jalan akses.

3.4 Operasi-operasi Dasar Aljabar Relasional

Untuk menetapkan bentuk dan batasan-batasan database, sebuah model data harus memasukkan sekumpulan operasi-operasi untuk mengontrol data. Sekumpulan model operasi-operasi relasional standar merupakan aljabar relasional. Operasi-operasi ini membolehkan user untuk menentukan dasar pencarian permintaan. Hasil dari sebuah pencarian adalah relasi yang baru, di mana mungkin

saja dibentuk dari satu atau lebih relasi. Oleh sebab itu, operasi-operasi aljabar menghasilkan relasi-relasi baru yang dapat menjadi lebih berguna dengan menggunakan operasi-operasi aljabar yang sama. Urutan dari operasi-operasi aljabar relasional membentuk sebuah ekspresi relasi aljabar yang hasilnya juga berupa sebuah relasi.

Operasi-operasi relasi aljabar umumnya dibagi ke dalam dua kelompok. Kelompok pertama, termasuk sekumpulan operasi-operasi dari sekumpulan teori matematika yang dapat dipakai karena tiap-tiap relasi ditetapkan menjadi sekumpulan tuple-tuple. Sekumpulan operasi-operasi tersebut termasuk *UNION, INTERSECTION, SET DIFFERENCE dan CARTESIAN PRODUCT*.

Kelompok yang kedua terdiri dari operasi-operasi yang khusus dibuat untuk database-database relasional. Yang termasuk dalam kelompok yang kedua adalah *SELECT, PROJECT dan JOIN*.

3.5 Operasi SELECT

Operasi SELECT digunakan untuk memilih sebuah subset tuple-tuple dari sebuah relasi yang memenuhi pilihan. Pada umumnya, operasi SELECT ditunjukkan oleh :

$$\sigma_{\text{<kondisi pilihan>}}(R)$$

dimana **"σ"** (sigma) digunakan untuk menetapkan operator SELECT dan <kondisi pilihan> adalah berupa ekspresi boolean yang ditetapkan pada attribute-attribute dari relasi R. Perlu diingat bahwa R pada

umumnya adalah sebuah ekspresi aljabar relasional yang hasilnya adalah sebuah relasi. Ekspresi yang paling sederhana dari operasi SELECT adalah sebuah nama dari sebuah relasi database. Relasi yang dihasilkan dari operasi SELECT mempunyai attribute yang sama, sebagai R. Ekspresi boolean yang ditetapkan dalam <kondisi pilihan> terbentuk dari sejumlah klausa-klausa dengan bentuk sebagai berikut :

<nama attribute><operator pembanding><nilai konstan>,

atau

<nama attribute><operator pembanding><nama attribute>

dimana <nama attribute> adalah nama sebuah attribute dari R, <operator pembanding> biasanya adalah salah satu dari operator-operator seperti "=","<",">=", "≤" dan "≠", dan <nilai konstan> adalah sebuah nilai konstan dari attribute domain. Klausa-klausa dapat mempunyai hubungan yang berbeda-beda dengan operator-operator boolean AND, OR, dan NOT untuk membentuk kondisi pilihan yang umum.

Sebagai contoh, dengan menggunakan tabel-tabel database pada lampiran A, untuk memilih tuple-tuple dari semua employee yang juga bekerja pada department 4 dan menghasilkan salary lebih dari $25.000 per-tahun, atau bekerja pada departement 5 dan menghasilkan $30.000 per-tahun, maka operasi SELECT yang dapat ditetapkan adalah :

$$\sigma_{\text{(DNO=4 AND SALARY>25000) OR (DNO=5 AND SALARY>30000)}}(\text{EMPLOYEE})$$

Umumnya, hasil dari sebuah operasi SELECT dapat ditentukan sebagai berikut. <selection condition> digunakan secara bebas untuk tiap-tiap tuple T dalam R. Hal ini dilakukan dengan mengganti tiap-tiap kejadian dari sebuah attribute A, dalam kondisi pilihan dengan nilainya pada tuple $t[A_i]$. Apabila kondisi dinilai benar, maka tuple t akan dipilih. Semua tuple-tuple yang dipilih akan tampak pada hasil dari operasi SELECT.

Operator SELECT adalah *unary*. Maka dari itu, operator SELECT digunakan untuk sebuah relasi tunggal. Lebih dari itu, operasi pemilihan digunakan untuk tiap-tiap tuple secara individu. Bahkan, kondisi-kondisi pilihan tidak dapat menggunakan lebih dari satu tuple. Jumlah tuple-tuple dari relasi yang dihasilkan selalu lebih kecil atau sama dengan jumlah tuple pada R. Maka dari itu, $|\sigma_c (R)| \leq |R|$ untuk setiap kondisi C. Bagian dari tuple-tuple yang dipilih oleh sebuah kondisi pilihan disebut sebagai **selectivity** dari kondisi.

3.6 Operasi PROJECT

Operasi PROJECT digunakan untuk memilih kolom-kolom tertentu dari tabel dan membuang kolom-kolom lainnya yang tidak diperlukan. Pada umumnya operasi PROJECT ditunjukkan oleh :

$\pi_{\text{<daftar attribute>}}$ **(R)**

di mana **"π"** (pi) adalah simbol yang digunakan untuk menggambarkan operasi PROJECT dan <daftar attribute> adalah

sebuah daftar attribute dari attribute-attribute pada relasi R. Perlu diingat bahwa R umumnya adalah ekspresi aljabar relasional yang hasilnya adalah relasi, dimana dalam kasus yang paling sederhana R adalah nama dari sebuah relasi database. Hasil dari operasi PROJECT hanyalah attribute-attribute yang ditentukan dalam <attribute list> dan dalam urutan yang sama seperti yang terlihat pada list.

3.7 Operasi JOIN

Operasi JOIN dilambangkan dengan **"⋈"** dan digunakan untuk mengkombinasikan hubungan tuple-tuple dari dua relasi kedalam tuple tunggal. Pada umumnya operasi PROJECT pada dua relasi $R(A_1,A_2,\ldots A_n)$ dan $S(B_1,B_2,\ldots B_m)$ ditunjukkan oleh :

R⋈ <kondisi join> (S)

Hasil dari JOIN adalah sebuah relasi Q dengan n + m attribute $Q(A_1,A_2,\ldots A_n, B_1,B_2,\ldots B_m)$. Q mempunyai satu tuple untuk masing-masing kombinasi dari tuple, satu dari R dan satu dari S. Dalam JOIN, hanya kombinasi-kombinasi dari tuple-tuple yang memenuhi kondisi join yang akan tampak pada hasil. Kondisi JOIN ditentukan oleh attribute-attribute dari relasi R dan S dan evaluasi untuk tiap kombinasi dari tuple-tuple.

Bentuk dari kondisi JOIN secara umum adalah :

<condition> AND <condition> AND … AND<condition>

dimana tiap kondisi adalah bentuk dari A_i θ B_j. A_i adalah sebuah attribute dari R dan B_j adalah sebuah attribute dari S. A_i dan B_j mempunyai domain yang sama, dan θ (theta) adalah salah satu dari operator-operator pembanding {<, ≤, ≠, ≥, >, =}. Operasi JOIN dengan sebuah kondisi join yang umum disebut dengan *theta join.*

3.8 Sekumpulan Operasi

Beberapa kumpulan operasi digunakan untuk menggabungkan elemen-elemen dari dua kelompok dalam cara yang bervariasi, termasuk *UNION, INTERSECTION dan SET DIFFERENCE*. Operasi-operasi ini adalah operasi-operasi binary yang masing-masing digunakan untuk dua kumpulan relasi.

Berikut definisi dari *UNION, INTERSECTION, dan SET DIFFERENCE* pada dua relasi R dan S .

- UNION : Hasil dari operasi ini ditunjukkan oleh R ∪ S, yaitu sebuah relasi yang merupakan semua tuple-tuple yang ada alam R atau S atau keduanya. Tuple duplikat dihilangkan.
- INTERSECTION : Hasil dari operasi ini ditunjukkan oleh R ∩ S, yaitu sebuah relasi yang merupakan semua tuple yang ada dalam R dan S.

- SET DIFFERENCE : Hasil dari operasi ini ditunjukkan oleh R – S, yaitu sebuah relasi yang merupakan semua tuple-tuple yang ada dalam R tetapi tidak ada dalam S.

Operasi CARTESIAN PRODUCT yang juga biasa disebut dengan CROSS PRODUCT atau CROSS JOIN dilambangkan dengan "X" yang juga merupakan sebuah kumpulan operasi binary. Operasi ini digunakan untuk mengkombinasikan tuple-tuple dari dua relasi dalam sebuah model gabungan. Umumnya, hasil dari R(A_1, A_2,…, A_n) X S(B_1, B_2, …, B_m) adalah relasi Q dengan n+m dari attribute-attribute R(A_1, A_2,…, A_n, B_1, B_2, …, B_m) dalam urutannya. Hasil dari relasi Q mempunyai satu tuple untuk masing-masing kondisi dari tuple-tuple. Satu dari R dan satu dari S.

3.9 Algoritma-algoritma untuk Menjalankan Operasi-operasi Query

Macam-macam operasi query seperti *Select, Join, Project* dan *set operation* (sekumpulan operasi) dapat diimplementasikan dengan menggunakan satu atau lebih *access routines* yang berbeda. *Access Routines* adalah algoritma-algortima yang digunakan untuk mengakses dan mengelompokkan data di dalam sebuah database.

Berikut ini akan dijelaskan mengenai algortima-algoritma yang digunakan untuk menjalankan operasi-operasi query di dalam sebuah sistem database. Algoritma-algoritma yang akan dijelaskan berikut ini antara lain adalah algoritma algoritma untuk mengimplementasikan operasi SELECT, algoritma-algoritma untuk mengimlementasikan operasi JOIN, algoritma untuk mengimplementasikan operasi PROJECT dan algoritma untuk mengimplementasikan *Sekumpulan operasi*.

3.9.1 Algoritma untuk Mengimplementasikan Operasi SELECT

Ada bermacam-macam metode untuk mengimplementasikan operasi SELECT. Dan pengimplementasian operasi select tergantung pada keberadaannya dan tipe pengindeks-annya serta kombinasi *conjunctive* dan *disjunctive* dari pencarian kriteria.

Pengimplementasian dari operasi SELECT dibagi lagi menjadi dua, yaitu implementasi untuk *simple selection* (pilihan sederhana) dan implementasi untuk *complex selection* (pilihan kompleks). Berikut ini akan dibahas mengenai algoritma untuk implentasi kedua pilihan tersebut.

3.9.1.1 Metode untuk Simple Selection

Sejumlah algoritma *search* memungkinkan untuk menyeleksi record-record dari sebuah file. Algoritma-algoritma search juga

dikenal sebagai *file scan* karena algoritma tersebut melakukan pembacaan record dari sebuah file untuk mencari dan mendapatkan kembali record-record yang memenuhi sebuah kondisi seleksi. Apabila algoritma search melibatkan penggunaan *index*, maka pencarian index disebut dengan *index scan*.

Berikut ini akan diberikan beberapa contoh dari *simple selection* yang data-datanya diambil dari file-file pada skema database COMPANY yang dapat dilihat pada lampiran A.

Contoh 1 : Salary > 28000 . [2]

Contoh 2 : SSN = ‘123456789’

Contoh 3 : DNO = 5

Contoh 1 dan contoh 3 diambil dari file EMPLOYEE. Sedangkan contoh 3 diambil dari file DEPARTMENT.

Simple selection dari contoh 1, contoh 2 dan contoh 3 dapat diimplementasikan dengan menggunakan metode-metode :

- ***Linear Search (brute force)***, untuk mendapatkan kembali setiap record dalam file dan menguji apakah nilai-nilai attribute dari record tersebut memenuhi kondisi pilihan.
- ***Binary Search***, apabila kondisi pilihan melibatkan sebuah perbandingan persamaan pada sebuah *key attribute* pada file yang berurutan. Maka binary search yang lebih efisien daripada linear search dapat digunakan. Contohnya adalah

SSN =’123456789’, apabila SSN adalah permintaan attribute untuk file EMPLOYEE.

- ***Primary Index atau Hash Key***, apabila kondisi pilihan melibatkan sebuah perbandingan persamaan pada sebuah key attribute dengan primary index (hash key). Sebagai contoh, SSN = ‘123456789’ menggunakan primary index untuk mendapatkan kembali record. Perlu diketahui bahwa kondisi ini kebanyakan mengembalikan sebuah record tunggal.
- ***Menggunakan primary index untuk mendapatkan kembali multiple record***, apabila perbandingan kondisi adalah >, >=, <, atau <= pada sebuah *key field* dengan sebuah primary index. Sebagai contoh, DNUMBER > 5 menggunakan index untuk menemukan record yang memenuhi persamaan kondisi yang cocok (DNUMBER = 5), kemudian mengembalikan semua record-record berikutnya pada urutan file. Untuk kondisi DNUMBER < 5, mengembalikan semua record-record terdahulu.
- ***Menggunakan clustering index untuk mendapatkan kembali multiple record***, apabila kondisi pilihan melibatkan sebuah perbandingan persamaan pada non-key attribute dengan sebuah clustering index. Sebagai contoh, DNO = 5 untuk file

[2] Disebut dengan batasan query

EMPLOYEE menggunakan index untuk mendapatkan kembali semua record yang memenuhi kondisi.

- ***Secondary (B^+-tree) index pada sebuah perbandingan persamaan***, untuk mendapatkan kembali sebuah record tunggal jika pengindeks-an field adalah sebuah *key* atau untuk mendapatkan kembali multiple record jika pengindeks-an field bukan sebuah key dan juga bisa digunakan untuk perbandingan-perbandingan yang meliputi >, >=, <, atau <=.

3.9.1.2 Metode untuk Complex Selection

Apabila sebuah kondisi dari operasi SELECT adalah sebuah kondisi konjungtif, yaitu terdiri dari beberapa kondisi-kondisi sederhana yang berhubungan dengan logika AND.

Berikut ini akan diberikan beberapa contoh dari *complex selection* yang data-datanya diambil dari file-file pada skema database COMPANY yang dapat dilihat pada lampiran A.

Contoh 4 : DNO = 5 AND SALARY > 30000 AND SEX = ‘F’

Contoh 5 : ESSN = ‘123456789’ AND PNO = 10

Contoh 4 diambil dari file EMPLOYEE. Sedangkan contoh 5 diambil dari file WORKS_ON.

Complex selection dari contoh 4 dan contoh 5 dapat diimplementasikan dengan menggunakan metode-metode :

- ***Conjunctive selection***. Menggunakan salah satu dari metode-metode akses dari simple selection untuk mendapatkan kembali record-record dari kriteria pertama yang cocok, kemudian menggunakan sekumpulan hasil untuk melakukan pengecekan dari sisa kriteria.
- ***Composite Index***. Apabila dua atau lebih attribute diperlukan dalam kondisi- kondisi persamaan pada kondisi konjungtif dan sebuah index gabungan tersedia pada field-field kombinasi. Sebagai contoh, jika sebuah index sudah dibentuk pada key gabungan (ESSN, PNO) dari file WORKS_ON untuk ESSN =’123456789’ AND PNO = 10, maka dapat digunakan index secara langsung.
- ***Intersection Record Pointers***. Apabila indeks-indeks secondary (jalan akses lainnya) tersedia pada lebih dari satu field yang diperlukan dalam kondisi-kondisi sederhana dalam kondisi konjungtif dan apabila indeks-indeks termasuk *pointer record*, kemudian tiap-tiap indeks dapat digunakan untuk mendapatkan kembali kumpulan pointer record yang memenuhi kondisi masing-masing. *Intersection* dari kumpulan-kumpulan pointer record ini memberikan pointer-pointer record yang memenuhi kondisi konjungtif, yang kemudian digunakan untuk mendapatkan kembali record-record tersebut secara langsung.

Apabila sebuah kondisi tunggal menentukan pilihan seperti contoh 1, contoh 2 dan contoh 3, maka yang dilakukan hanyalah mengecek apakah sebuah jalan akses tersedia pada attribute yang termasuk dalam kondisi tersebut. Apabila sebuah jalan akses tersedia, maka metode yang cocok untuk jalan akses tersebut digunakan. Sebaliknya, pendekatan metode brute force linear search dapat digunakan. Optimisasi query untuk sebuah operasi SELECT kebanyakan diperlukan untuk kondisi-kondisi pilihan konjungtif apabila lebih dari satu attribute terlibat dalam kondisi-kondisi yang mempunyai sebuah jalan akses. Optimizer harus memilih jalan akses yang mendapatkan record-record paling sedikit dengan cara yang paling efisien dengan memperkirakan harga-harga yang berbeda dan memilih metode dengan perkiraan harga yang paling sedikit.

Pada saat optimizer memilih antara kondisi-kondisi multiple yang sederhana dalam sebuah kondisi pilihan konjungtif, maka optimizer secara khusus mempertimbangkan *selectivity* dari masing-masing kondisi. ***Selectivity* (s)** didefinisikan sebagai perbandingan dari nomer record-record (tuple-tuple) yang memenuhi kondisi untuk jumlah nomer dari record-record pada file (relasi). Yaitu sebuah nomer antara 0 (nol) dan 1 (satu), yang berarti selectivity 0 (nol) berarti tidak ada record yang memenuhi kondisi dan 1 (satu) berarti semua record memenuhi kondisi. Meskipun selectivity-selectivity yang tepat dari semua kondisi-kondisi mungkin tidak tersedia,

perkiraan dari selectivity-selectivity tersebut seringkali disimpan dalam katalog DBMS dan digunakan oleh optimizer. Sebagai contoh, untuk sebuah persamaan kondisi pada key attribute dari relasi r(R), s= 1/ |r(R)|, di mana |r(R)|adalah nomer dari tuple-tuple dalam relasi r(R). Untuk sebuah kondisi persamaan pada sebuah attribute dengan *i distinct values*, s diperkirakan oleh (|r(R)|/ *i*) /|r(R)| atau 1/*i* dengan anggapan bahwa record-record dibagi diantara distinct value. Menurut anggapan tersebut, record-record |r(R)|/ *i* akan memenuhi sebuah kondisi persamaan pada attribute ini. Umumnya, nomer dari record-record yang memenuhi sebuah kondisi pilihan dengan selectivity s akan diperkirakan menjadi |r(R)| * s. Perkiraan yang lebih kecil ini adalah memerlukan penggunaan kondisi pertama yang lebih tinggi untuk mendapatkan kembali record-record.

Perbandingan untuk sebuah kondisi pilihan konjungtif adalah sebuah kondisi *disjunctive* (dimana kondisi-kondisi sederhana dihubungkan oleh logika OR lebih daripada AND) yang lebih sulit untuk diproses dan dioptimisasi. Sebagai contoh adalah :

σDNO = 5 OR SALARY > 30000 OR SEX = 'F'

Dengan sebuah kondisi yang demikian, optimisasi yang sederhana dapat dilakukan, karena record-record yang memenuhi kondisi disjungtif adalah *union* (penggabungan) dari record-record yang memenuhi kondisi-kondisi individual. Malahan, apabila terdapat salah satu dari kondisi tersebut tidak mempunyai sebuah jalan akses,

maka terpaksa menggunakan pendekatan brute force linear search. Hanya jika sebuah jalan ada pada setiap kondisi pilihan yang dapat dioptimisasi dengan mendapatkan kembali record-record yang memenuhi tiap-tiap kondisi atau record id-nya, dan kemudian menentukan operasi union untuk mengeleminasi kondisi-kondisi yang sama.

Sebuah DBMS menyediakan beberapa metode yang sudah didiskusikan sebelumnya dan khususnya beberapa metode-metode tambahan. Query optimizer harus memilih salah satu metode yang tepat untuk mengeksekusi tiap-tiap operasi SELECT dalam sebuah query. Pada akhirnya, optimizer akan memilih metode akses dengan perkiraan harga terendah.

3.9.2 Algoritma untuk Mengimplementasikan Operasi JOIN

Operasi JOIN merupakan salah satu dari kebanyakan operasi-operasi lainnya yang memakan waktu dalam melakukan pemrosesan query. Algoritma–algoritma yang dipertimbangkan untuk bentuk operasi JOIN adalah R $\bowtie_{A=B}$ S di mana A dan B adalah attribute-attribute domain yang cocok dari R dan S, secara berturut-turut. Sejumlah metode yang tersedia untuk mengimplementasikan operasi JOIN adalah sebagai berikut :

- ***Nested Loop (brute force)***. Untuk setiap record t pada R (outer loop), dapatkan kembali record s dari S (inner loop) dan

lakukan tes untuk dua record yang memenuhi kondisi JOIN t[A] = s[B]. [4]

- ***Access Structures (Indexes)* atau *single-loop join***. Jika sebuah indeks (hash key) tersedia untuk satu atau dua attribute join (B dari S), dapatkan kembali masing-masing record t dalam R, sekali pada satu saat (single loop) dan kemudian dengan menggunakan akses untuk mendapatkan kembali semua record-record s dari S yang memenuhi s[B] = t[A].
- ***Sort-Merge Join***. Jika record-record dari R dan S diurutkan secara fisik oleh nilai dari attribute-attribute join dari A dan B berturut-turut, maka join dapat diimplementasikan dalam banyak cara yang memungkinkan dan efisien. Kedua file tersebut dibaca secara bersamaan dalam perintah join attribute, mencocokkan record-record yang mempunyai nilai-nilai yang sama untuk A dan B. Apabila file-file tersebut belum urut, maka akan diurutkan terlebih dahulu dengan menggunakan *eksternal sorting*.[5]
- ***Hash Join***. Record-record dari file-file R dan S, keduanya digabungkan pada file hash yang sama, menggunakan fungsi-fungsi hash yang sama pada attribute-attribute join A dari R

[4] Untuk file-file disk, jelas bahwa pengulangan akan memenuhi blok-blok disk. Jadi teknik ini dapat juga disebut dengan *nested-block join*.
[5] Merupakan algoritma sorting yang cocok untuk file-file yang besar dari penyimpanan record pada disk.

dan B dari S sebagai hash key. Mula-mula, sebuah single pass melalui file yang dengan record yang lebih sedikit (dimisalkan dengan R) menggabungkan record-recordnya ke dalam file hash selama record-record R dibagi ke dalam bucket-bucket hash. Pada tahap yang kedua, sebuah single pass melalui file lainnya (S) kemudian menggabungkan tiap-tiap record-nya untuk memeriksa bucket yang cocok, dan record tersebut dikombinasikan dengan semua record-record yang cocock dari R dalam bucket tersebut. Penyederhanaan ini menggambarkan bahwa ukuran yang lebih kecil dari dua file dimasukkan ke dalam bucket-bucket memory secara tepat setelah melalui tahap pertama.

3.9.3 Algoritma untuk Mengimplementasikan Operasi PROJECT dan Sekumpulan Operasi

Sebuah operasi PROJECT $\pi_{<\text{attribute list}>}(R)$ akan dilaksanakan apabila <attribute list> termasuk sebuah key dari relasi R, karena dalam kasus ini hasil dari operasi PROJECT akan mempunyai nomer dari tuple-tuple yang sama sebagai R, tetapi hanya dengan nilai-nilai untuk attribute-attribute dalam <attribute list> pada tiap tuple. Apabila <attribute list> tidak termasuk sebuah key dari R, maka tuple-tuple yang sama harus dieliminasi. Pengeliminasian ini biasanya dilakukan dengan melakukan sorting pada hasil operasi dan kemudian

mengeliminasi tuple-tuple yang sama, yang tampak secara berurutan setelah sorting. Hashing juga dapat digunakan untuk mengeliminasi record-record yang sama dengan cara masing-masing record dibagi dan diselipkan ke dalam sebuah bucket pada hash file dalam memory. Dan kemudian dilakukan pengecekan kembali untuk memastikan apakah record-record tersebut sudah berada di dalam bucket. Apabila record yang dimasukkan tersebut merupakan record yang sama, maka record tersebut tidak diselipkan ke dalam bucket.

Sekumpulan operasi seperti UNION, INTERSECTION, SET DIFFERENCE dan CARTESIAN PRODUCT kadang-kadang sulit untuk dilakukan.Umumnya, operasi cartesian product R X S lebih sulit untuk dilakukan karena hasil dari operasi ini adalah sebuah record untuk tiap-tiap kombinasi dari record-record R dan S. Apabila R mempunyai n record dan j attribute dan S mempunyai m record dan k attribute, maka hasil dari relasi kedua record tersebut adalah n * m record dan j + k attribute. Oleh sebab itu, operasi CARTESIAN PRODUCT harus dihindari dan digantikan dengan operasi-operasi lainnya yang sama selama optimisasi query.

Tiga kumpulan operasi-operasi lainnya seperti UNION, INTERSECTION, dan SET DIFFERENCE hanya dipakai untuk relasi-relasi union yang cocok, yang mempunyai nomer attribute yang sama dan domain-domain attribute yang sama. Cara yang biasanya digunakan untuk melaksanakan operasi-operasi ini adalah dengan

menggunakan variasi-variasi dari teknik *sort-merge*, yaitu dua relasi diurutkan pada attribute yang sama dan setelah diurutkan dilakukan sekali pembacaan pada tiap-tiap relasi yang memenuhi untuk menghasilkan hasilnya. Sebagai contoh, operasi UNION dapat dilaksanakan (R ∪ S) , dengan *scanning* (pembacaan) dan *merging* (penggabungan) kedua file-file yang sudah urut secara bersamaan dan apabila tuple yang sama tersedia dalam kedua relasi, maka dipilih hanya satu tuple untuk disimpan dalam hasil penggabungan. Untuk operasi INTERSECTION, R ∩ S, maka akan disimpan pada hasil gabungan hanya untuk record-record yang tampak pada kedua relasi.

Hasing juga dapat digunakan untuk melaksanakan UNION, INTERSECTION, dan SET DIFFERENCE. Satu tabel dibagi dan yang lainnya digunakan untuk memeriksa partisi yang tepat. Sebagai contoh, untuk melaksanakan R ∪ S, maka pertama-tama partisi record-record dari R untuk hash file. Kemudian bagi tiap-tiap record S, dilakukan pemeriksaan untuk mengecek apabila sebuah record yang sama dari R ditemukan dalam bucket.

BAB IV
PERINTAH SQL

SQL *(Structured Query Language)* merupakan bahasa non-prosedural yang tidak menyediakan struktur pemrograman tradisional. Tetapi SQL mampu menentukan operasi yang ingin Anda jalankan pada tingkat tinggi yang perincian dari penerapannya bisa diserahkan pada DBMS. DBMS yang akan mengambil dan menampilkan baris-baris dari informasi yang pengguna minta. Dengan SQL, tidak perlu menentukan bagaimana operasi pemilihan baris berlangsung, yang diperlukan hanya perlu menentukan apa yang harus dilakukan oleh database untuk pengguna (bukan bagaimana cara melakukannya).

Pernyataan SQL dapat disusun dengan operasi mouse yang disebut QBE (*Query by Example*) seperti yang terdapat pada Microsoft Access dan Microsoft SQL Server, tetapi semuanya itu tidak bisa menggantikan penulisan detail pernyataan SQL yang kompleks.

Dan SQL sendiri tidak memiliki interface, sehingga untuk membuat form tampilan dibutuhkan bantuan program lain seperti Visual Basic.

Pernyataan SQL dikategorikan ke dalam dua kategori utama, yaitu:

1. DML (*Data Manipulation Language*); pernyataan untuk memanipulasi data, seperti mengambil, menyisipkan data baru, menyunting atau menghapus data yang sudah ada.
2. DDL(*Data Definition Language*); pernyataan untuk mendefinisikan suatu object database, yang ini bukan bidang dari pengembang database.

Selection Query dan Action Query

Pernyataan pada banyak DML dari bahasa SQL juga dikenal dengan istilah *query*. Ada dua jenis query yaitu

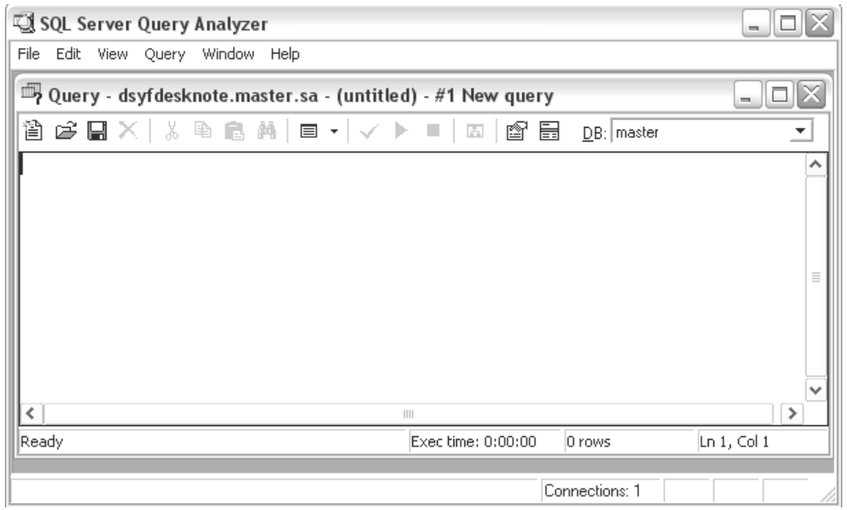
1. *Selection Query ;* mengambil informasi dan tidak memodifikasi database. Pernyataan yang digunakan adalah SELECT
2. *Action Query*; dapat memodifikasi data pada tabel-tabel database dan diawali dengan salah satu pernyataan berikut : INSERT, SELECT, atau UPDATE.

4.1 Menjalankan Pernyataan SQL

Untuk belajar SQL, dapat menggunakan beberapa program yang mendukung operasi SQL seperti *Query Analyzer* (Microsoft SQL Server), *QBE (* Microsoft Access), dsb.

4.2 Query Analyzer

Jalankan *Query Analayzer* (pilih Start> Microsoft SQL Server > Query Analyzer), pada awalnya jendela ini akan kosong seperti gambar berikut :



Gambar 4.1. Jendela Penulisan Query

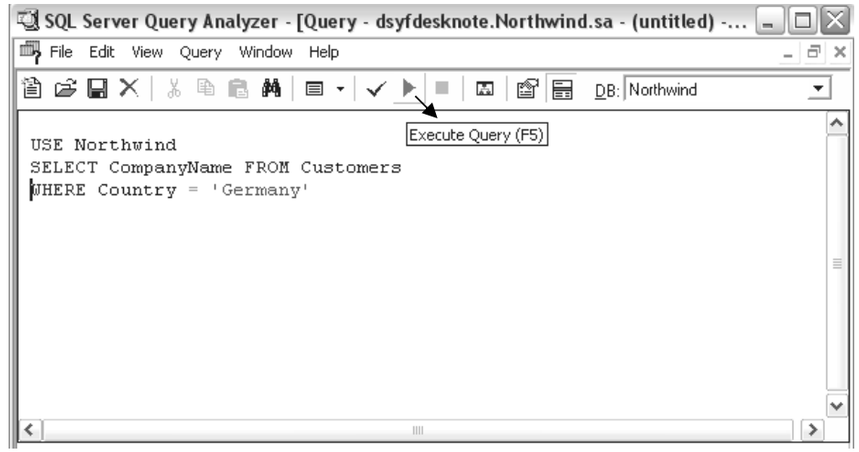
Cara menulis pernyataan :

1. Ketikkanlah pernyataan SQL yang ingin dijalankan, lalu pilihlah nama database yang diinginkan pada daftar DB. Pernyataan SQL tersebut akan dijalankan pada database yang dipilih.

2. Awali pernyataan SQL dengan pernyataan USE, yang menentukan database yang akan dimanipulasi. Misalnya untuk mengambil informasi pelanggan Northwind yang terletak di Germany, ketik pernyataan berikut ini :

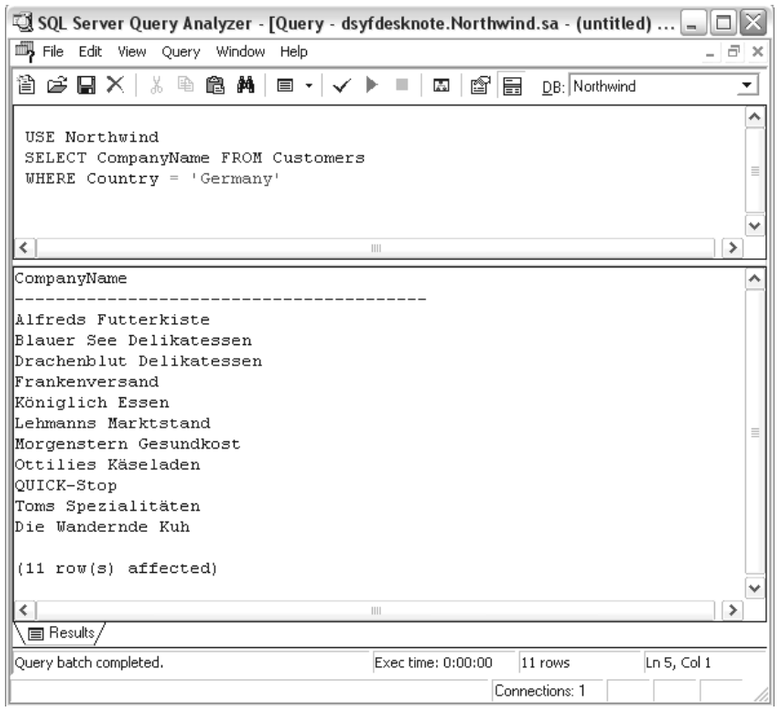
```
USE Northwind
SELECT CompanyName FROM Customers
WHERE Country = 'Germany'
```

Lalu pilihlah perintah *Execute* dari menu Query, atau tekan Ctrl+E untuk menjalankan pernyataan tersebut.



Gambar 4.2. Penyuntingan Query

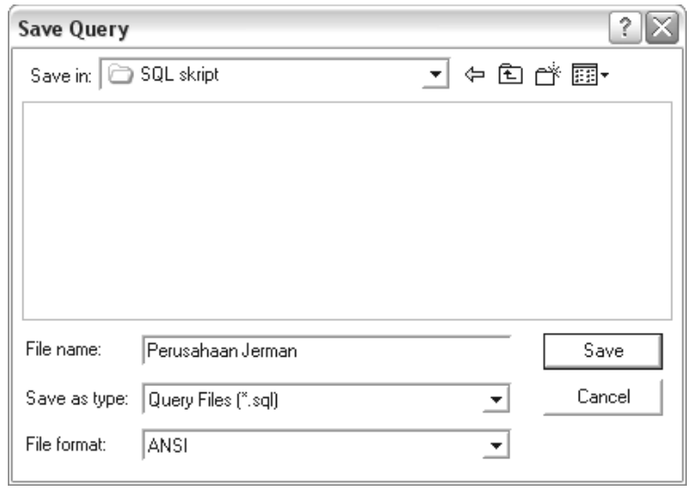
Hasilnya akan muncul pada panel di bagian bawah, seperti ditunjukkan pada gambar berikut :



Gambar 4.3. Hasil eksekusi Query

Lihat baris-baris (record) yang dipilih serta jumlahnya pada bagian bawah panel *result*.

Untuk menjalankan Query lain, ketikkanlah pernyataan lain di panel atas, atau suntinglah pernyataan sebelumnya, lalu tekan Ctrl+E kembali. Bisa pula untuk menyimpan pernyataan SQL ke dalam file, agar tidak perlu mengetikkannya kembali. Untuk melakukannya, bukalah menu *File*, pilih *Save* atau *Save as*, dan ketikkan nama file dimana isi dari panel Query akan disimpan.

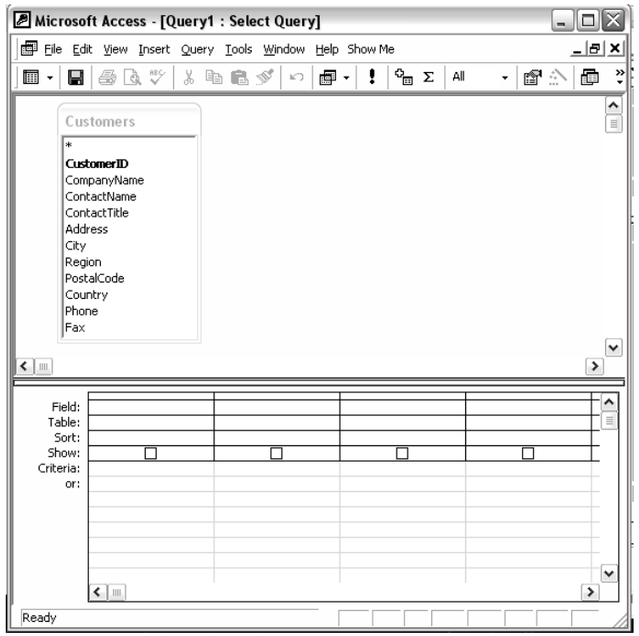


Gambar 4. 4 Penyimpanan Query

Pernyataan tersebut akan disimpan sebagai file teks dengan ekstensi .**sql**.

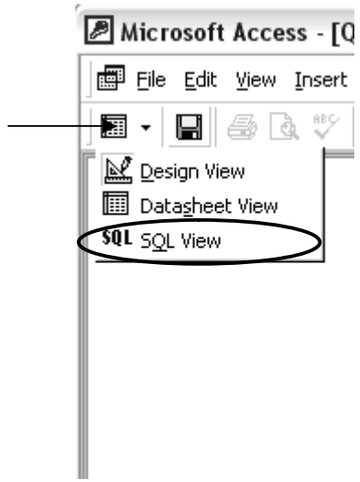
4.3 Query Access

Selain *Microsoft SQL Server* dapat menggunakan *Microsoft Access* yang lebih familiar. Dalam membuat Query baru, pilihlah tab *Queries* dari databases, dan tekan tombol *New*. Untuk menyunting query yang sudah ada, tekanlah tombol *Design*. Apabila membuat query baru, Access akan menampilkan jendela *Query by Example* (kalau di *Microsoft SQL, QBE* menggunakan *Enterprise Manager)*. Sarana ini mengizinkan membuat query dengan operasi mouse saja.



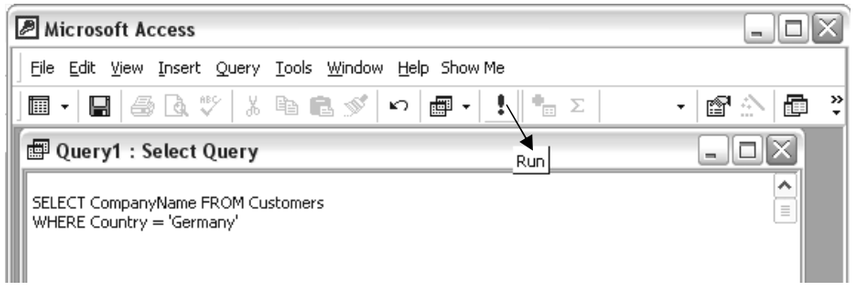
Gambar 4.5. Query by Example pada Access

Untuk menulis pernyataan SQL, harus menutup mode *Query by Example* ini dan berpindah ke penyunting teks kecil di mana bisa mengetikkan pernyataan.



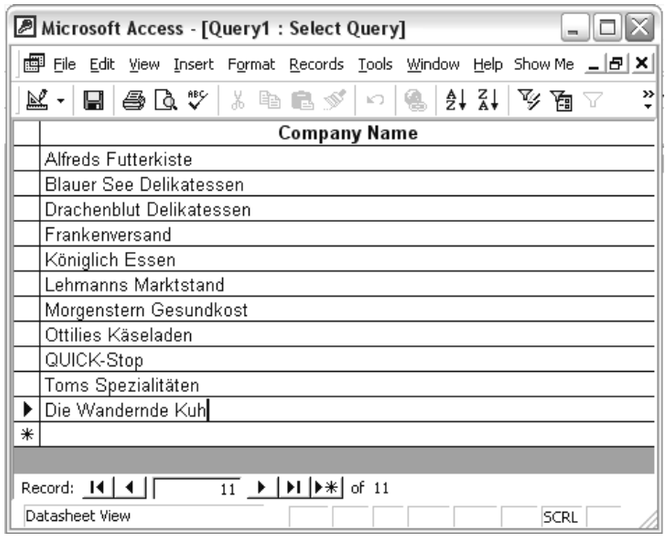
Gambar 4.6. Pilih SQL View

Ketikkan pernyataan SQL, seperti ditunjukkan pada gambar 4.7, kemudian eksekusi pernyataan query tersebut.



Gambar 4.7. Penyuntingan Pernyataan Query

Hasilnya adalah sebagai berikut :



Gambar 4.8. Hasil Eksekusi

Untuk menyimpannya juga berupa query dengan memilih menu file lalu *Save as* kemudian masukkan nama dari query baru tersebut.

Tidak seperti SQL server, Access mengijinkan menyunting hasil dari Query, selama perubahan yang dibuat itu sah dan tidak melanggar integritas dari database.

4.4 Data Manipulation Language

Data Manipulation Language terdiri atas empat buah pernyataan, seperti dijelaskan pada tabel berikut :

Tabel 4.1 Pernyataan DML

No	Pernyataan	Tindakan
1	SELECT	Mengambil record-record dari database
2	INSERT	Menambahkan record baru ke database
3	DELETE	Menghapus record dari database
4	UPDATE	Mengganti record di database

Pernyataan-pernyataan di atas bisa menjadi rumit, karena harus dapat mendefinisikan record-record mana saja yang harus dipilih pada banyak tabel, dan SQL menyediakan banyak operator dan fungsi yang bisa digunakan untuk menentukan record yang diinginkan.

Mekanisme untuk menentukan baris yang diinginkan adalah sama, baik melakukan pemilihan, mengupdate, atau menghapus record.

Pernyataan INSERT adalah yang paling sederhana, karena ia hanya menambahkan baris ke dalam tabel pada database.

4.5 Pernyataan SELECT

Bentuk sederhana pernyataan SELECT adalah

```
SELECT  fields
FROM   tables
```

Di mana *fields* dan *tables* adalah daftar yang dipisahkan dengan koma yang terdiri atas field-field (kolom-kolom) yang ingin diambil dari tabel suatu database.

Contoh :

1. Untuk memilih informasi kontak dari semua perusahaan pada tabel Customers, gunakanlah pernyataan berikut ini :

```
SELECT  CompanyName, ContactName
FROM  Customers
```

2. Untuk mengambil semua field, gunakanlah tanda asterisk (*) atau kata kunci ALL :

```
SELECT * FROM  Customers
```


4.5.1 Klausa WHERE

Untuk mengambil kolom-kolom tertentu dari semua baris pada tabel, biasanya harus menentukan kriteria tertentu seperti, “Semua perusahaan di Germany” , “Semua pelanggan yang telah memesan tiga atau lebih pesanan dalam enam bulan terakhir,” atau ekspresi yang jauh lebih kompleks lagi. Untuk membatasi jumlah baris yang dihasilkan oleh query, gunakanlah klausa WHERE pada pernyataan SELECT. Bentuk yang paling umum dari pernyataan WHERE adalah seperti berikut ini :

```
SELECT   fields
FROM   tables
WHERE   condition
```

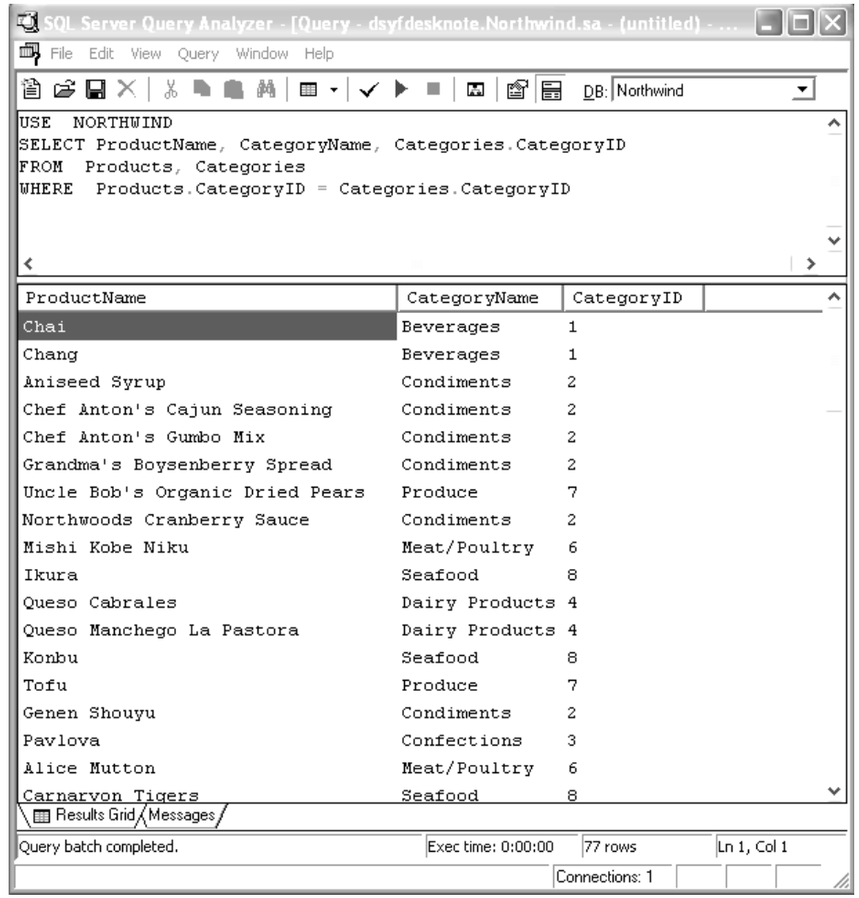
Argumen *condition* bisa merupakan ekspresi relasional, seperti yang Anda gunakan pada VB. Untuk memilih semua pelanggan dari Germany, gunakanlah kondisi berikut ini :

```
WHERE  Country   =  ‘Germany’      OR
      Country    =   ‘Australia’    OR
      Country    =   ‘Switzerland’
```

4. 5.2 Menggunakan Banyak Tabel

Apabila menggabungkan banyak tabel di dalam query, harus selalu menyertakan klausa WHERE untuk menentukan beberapa kriteria. Misalkan ingin menampilkan daftar semua nama produk bersamaan dengan kategorinya. Informasi yang dibutuhkan tidak terdapat dalam satu tabel, karena harus mengambil nama produk dari tabel Products dan nama kategori dari tabel Categories dan menentukan bahwa field ProductID pada kedua tabel harus sama. Pernyataan selengkapnya sebagai berikut :

```
USE  NORTHWIND
SELECT ProductName, CategoryName
FROM  Products, Categories
WHERE  Products. CategoryID = Categories.CategoryID
```

Gambar 4.9. Menggunakan Banyak Tabel

Pernyataan ini mengambil nama-nama dari semua produk, bersamaan dengan kategorinya. Beginilah cara kerja pernyataan ini saat dijalankan: untuk setiap baris pada tabel Products, *SQL engine* akan mencari baris yang bersesuaian pada tabel Categories, lalu menambahkan field ProductName CategoryName, dan CategoryID ke dalam hasilnya. Baris-baris tersebut dicocokkan dengan field CategoryID (kedua baris harus memiliki nilai CategoryID yang sama). Jika sebuah produk tidak memiliki kategori, produk tersebut tidak akan ditampilkan pada hasil.

Perhatikan apabila field-field pada dua tabel yang berbeda memiliki nama yang sama, misalnya pada CategoryID sendiri harus mengawalinya dengan nama tabel untuk menghilangkan kerancuan (ambiguos).

Perhatikan juga bahwa beberapa nama field yang mengandung spasi, mama field harus ditulis dalam tanda kurung siku […].

Anda juga bisa menggabungkan beberapa batasan dengan operator logikal. Untuk mengambil semua judul buku yang diterbitkan oleh penerbit tertentu, gunakanlah nomor ID penerbit pada klausa WHERE, seperti pada pernyataan berikut ini. Kali ini saya menggunakan database Biblio Access, karena ia memiliki lebih banyak data daripada database Pubs :


```
USE Biblio
SELECT  Titles.Title
FROM Titles
WHERE Title.pubID=722
```

Pernyataan ini menganggap Anda sudah tau nomor ID penerbit (Atau Anda sebelumnya melihatnya pada tabel Publisher dan memasukkannya ke dalam pernyataan). Kode program yang lebih baik dari pernyataan yang sama adalah seperti berikut ini :

```
SELECT Titles.Title
FROM Titles.Publishers
WHERE Titles.PubID=Publishers.PubID AND
      Publishers.Name=”Sybex”
```

Pernyataan ini menggabungkan dua tabel dan memilih judul buku dari penerbit yang ditentukan berdasarkan namanya. Untuk mencocokkan judul buku dan penerbitnya, pernyataan ini meminta:

1. Nama penerbit pada tabel Publishers adalah SYBEX, dan
2. Field PubID pada tabel Title sesuai dengan field PubID pada tabel Publishers.

Perhatikan bahwa kita tidak menentukan nama penerbit (field [Company Name]) pada daftar SELECT. Semua buku yang

diinginkan memiliki penerbit yang sama, jadi kita menggunakan field PubID pada klausa WHERE, tetapi tidak menyertakan informasi apapun mengenai penerbit pada hasilnya.

Jika Anda menentukan banyak tabel tanpa klausa WHERE, pernyataan SQL akan menghasilkan cursor yang besar. Jika Anda mengetikkan pernyataan berikut ini :

```
SELECT ProductName, CategoryName FROM Categories, Products
```

Anda tidak akan mendapatkan satu baris untuk setiap nama produk yang diikuti oleh kategorinya. Tetapi Anda akan mendapatkan sebuah cursor dengan 616 baris, yang merupakan semua kombinasi yang mungkin dari nama-nama produk serta nama-nama kategori. Ini disebut dengan Cartesian Product 9atau cross-product) dari kedua tabel dan mengandung semua kombinasi yang mungkin dari baris-baris dari kedua tabel. Pada contoh ini, tabel Categories memiliki delapan baris dan tabel products memiliki 77 baris, jadi cross-product kedua tabel ini akan mengandung 616 baris. Jika Anda membuat cross-product dari dua tabel yang mengandung ribuan baris, Anda akan mendapatkan sebuah cursor yang begitu besar yang biasanya akan sia-sia saja.

4. 5.3 Kata Kunci AS

Secara default, setiap kolom pada query diberi nama seperti nama field sesungguhnya pada output. Jika sebuah tabel mengandung dua field bernama CustLName dan CustFName, Anda bisa menampilkannya dengan label yang berbeda dengan menggunakan kata kunci AS. Pernyataan SELECT berikut ini :

```
SELECT CustLName, CustFName
```

Akan menghasilkan dua kolom berlabel CustLName dan CustFName. Output dari query akan jauh lebih baik jika Anda mengubah label kedua kolom ini dengan pernyataan seperti berikut ini :

```
SELECT CustLName AS [Last Name], CustFName AS [First Name]
```

Anda juga bisa menggabungkan kedua field pada daftar SELECT dengan opartor penggabungan. Field-field yang digabungkan tidak akan diberi label secara otomatis, jadi Anda harus menentukan header Anda sendiri untuk field yang digabungkan tersebut. Pernyataan berikut ini akan membuat sebuah kolom tunggal untuk nama pelanggan dan memberinya nama customer Name:

```
SELECT  CustFName + “ , “ + CustLName AS [Customer Name]
```

SQL Server secara tidak otomatis memberi label pada kolom yang digabungkan. Access memberinya nama Expr1000, Expr1001, dan seterusnya. Perhatikan pula bahwa Access mengenali kedua operator untuk penggabungan yaitu & dan +. Operator dan penggabungan kedua nilai, bahkan jika salah satu (atau keduanya) bersifat numerik. Operator + menggabungkan kedua argumen jika keduanya bertipe string, tetapi menjumlahkannya apabila keduanya bersifat numerik. SQL Server mengenali satu buah operator penggabungan saja (simbol +). Jika kedua nilai yang digabungkan bertipe numerik, nilai tersebut akan dijumlahkan. Jika keduanya bertipe teks, akan digabungkan. Jika salah satunya bertipe numerik, SQL Server tidak akan melakukan pengubahan secara langsung. Anda harus mengubah nilai numerik tersebut menjadi teks dengan fungsi CONVERT () sebelum bisa menggabungkan keduanya dengan operator

4. 5.4 Kata Kunci TOP

Beberapa query mungkin akan mengambil baris dalam jumlah besar, padahal anda hanya menginginkan beberapa baris paling atas saja. Kata kunci TOP N mengizinkan Anda memilih N baris pertama dan mengabaikan baris-baris berikutnya. Kita misalkan Anda ingin melihat daftar dari 10 produk paling laku. Tanpa kata kunci TOP, Anda harus menghitung berapa banyak item dari setiap produk yang

telah dijual, menurutkannya berdasarkan item yang terjual, dan memeriksa 10 baris pertama yang dihasilkan oleh query.

Kata kunci TOP hanya digunakan apabila baris-baris diurutkan berdasarkan kriteria yang berarti. Membatasi output query secara alfabetis untuk mendapatkan N baris teratas tidaklah praktis. Apabila baris-baris diurutkan berdasarkan item yang terjual, penghasilkan yang dihasilkan, dan sebagainya, kita perlu membatasi output query ke dalam N baris.

4. 5.5 Operator LIKE

Seringkali Anda menginginkan hasil-hasil yang tidak benar-benar tepat. Nama penerbit mungkin diikuti dengan beberapa inisial, misalnya Inc, SA, dan sebagainya. Atau Anda mungkin ingin mengambil buku-buku dengan kata tertentu pada judulnya. Untuk menangani pencarian seperti ini, yang melibatkan pola daripada kecocokan yang tepat, SQL menyediakan operator LIKE, yang merupakan operator pencocokan pola yang sangat umum.

Operator LIKE menggunakan karakter pencocokan pola, seperti yang Anda gunakan untuk memilih banyak file pada DOS. Ekspresi “Sales*” memiliki arti “Semua file yang diawali dengan string Sales,” sementara ekspresi “*.xls” berarti “Semua file yang memiliki ekstensi .xls” Operator LIKE mengenali sejumlah karakter pencocokan pola (atau karakter wildcard) untuk mencocokkan satu

atau lebih karakter, angka numerik, huruf dan sebagainya. Masalahnya, karakter wildcard tidak identik pada semua DMBS, tapi kita akan membahas ini nanti.

Jika Anda tidak ingat nama belakang suatu penerbit, gunakanlah ekspresi seperti berikut ini :

```
WHERE Publishers. Name LIKE “John Wiley”
```

Ekspresi ini akan mencari semua baris pada tabel Publishers yang diawali dengan string “JOHN WILEY” dan diikuti oleh semua karakter lain (atau tanpa karakter sama sekali). Pencarian ini tidak mempedulikan kapitalisasi, jadi Anda tidak perlu mengubah kapitalisasi argumennya. Jika Anda ingin menampilkan semua buku yang diterbitkan oleh John Wiley pada database Biblio dengan menggunakan operator sama dengan (Publishers.Name = “JOHN WILEY”), Anda hanya akan mendapatkan dua record. Jika Anda menggunakan operator LIKE, seperti ditunjukkan pada pernyataan SQL berikut ini, Anda akan mendapatkan 394 judul buku :

```
SELECT  Titles.Title
FROM  Titles.Publishers
WHERE  Titles.PubID=Publishers.PubID AND
                Publishers.Name LIKE “JOHN WILEY”
```


Record-record lainnya merupakan judul buku yang diterbitkan oleh John Wiley & Sons. Anda bisa menggunakan operator LIKE untuk mengambil semua judul buku tentang windows, dengan pernyataan seperti berikut ini :

```
SELECT   Titles.Title
FROM   Titles
WHERE   Titles.Title LIKE “*WINDOWS*” AND
                Titles.Title LIKE “*PROGRAM*”
```

Dengan menggunakan String “*PROGRAM*” Anda bisa mencari judul buku dengan kata Program, Programming, Programmer, dan sebagainya. Kumpulan buku ini kemungkinan besar akan mengandung semua judul buku tentang pemrograman Windows daripada jika Anda menggunakan string “PROGRAMMING” pada klausa WHERE.

Operator LIKE mengenali banyak karakter wildcard, tetapi sintaksnya tidak sama pada semua database. Bahkan Microsoft Access dan SQL Server menggunakan karakter wildcard yang berbeda. Jika anda menggunakan satu produk saja, Anda akan terbiasa dengan notasinya dengan cepat. Jika Anda menjadi konsultan untuk banyak perusahaan, sintaks dari operator LIKE bisa menjadi masalah.

Karakter wildcard Access sama seperti yang digunakan pada DOS dan WINDOWS untuk pencocokan file dan direktori (seperti *.exe atau Sales??.xls).

Untuk menggunakan simbol khusus sebagai literal, Anda harus menutupinya dengan tanda kurung siku. Anda bisa mencari tanda kurung buka, tanda tanya, simbol matematis, dan arsitek sebagai literal pada argumen pencarian dengan menuliskannya di dalam tanda kurung siku. Tanda kurung tutup tidak bisa ditampilkan di dalam pasanganj tanda kurung siku (ia akan menutup tanda kurung buka lebih awal), tetapi Anda bisa menggunakannya sebagai literal sendiri. Tanda kurung tutup pada argumen “adc]” tidak memiliki arti, karena tidak ada tanda kurung buka.

Apabila melakukan perbandingan string dengan LIKE, semua karakter pada string akan diperhatikan, termasuk spasi di depan atau di akhir karakter. Untuk mengabaikan spasi di dalam argumen pencarian, gunakanlah fungsi TRIM () pada Access, atau LTRIM () dan RTRIM () pada SQL Server. LTRIM () dan RTRIM () berfungsi menghilangkan spasi di depan dan di belakang karakter. Kesemua fungsi ini bukanlah bagian dari SQL, tetapi SQL Server maupun Access mengenali fungsi-fungsi untuk menghilangkan spasi pada string. Situasi sulit yang lain mungkin muncul karena spasi bisa terdapat di dalam nilai field. Field yang dideklarasikan sebagai char

(20) akan mengandung spasi apabila nilai yang disimpan di dalamnya panjangnya kurang dari 20 karakter. Semua field yang menyimpan string dengan panjang bervariasi harus dideklarasikan sebagai varchar.

Beberapa argumen pencarian mungkin mengandung satu atau lebih karakter wildcard khusus. Sebuah kolom yang menyimpan diskon bisa mengandung nilai numerik diikuti oleh karakter persen (%), dan karakter garis bawah sudah sering pula digunakan pada kode produk. Untuk menyisipkan karakter wildcard di dalam argumen pencarian sebagai karakter yang harus dicari, Anda harus mkengetikkan karakter tersebut dua kali. Sebagai contoh, untuk mencari string “Up to 50 % off the list price,” gunakanlah klausa WHERE berikut ini :

```
WHERE Comments LIKE ‘%up to 50 %% off the list price%’
```

Bagaimana jika Anda ingin mencari string “50%”? Anda ingin mencari sebuah string yang diakhiri dengan tanda persen, diikuti oleh karakter apapun. Kali ini Anda harus menggunakan klausa ESCAPE. Klausa ini menentukan karakter escape, yang membatalkan efek dari karakter khusus, apabila diletakkan di depannya. Untuk mencari yang mengandung string ‘50’, Anda perlu menggunakan klausa WHERE seperti berikut ini :

```
WHERE Comments LIKE '%50%'
```

Tetapi jika Anda ingin mencari string "50%", Anda sebelumnya harus memilih sebuah karakter escape, yang tidak terdapat pada argumen pencarian (tanda seru biasanya digunakan sebagai karakter escape). Kondisi harus diubah menjadi '%50!%%' dan diikuti oleh klausa ESCAPE :

```
WHERE Comments LIKE '50!%%' ESCAPE '!'
```

Tanda persen pertama dan terakhir pada argumen pencarian memiliki arti yang biasa (semua karakter di depan atau di belakang string). Tanda seru memerintahkan agar SQL Server menganggap tanda persen berikutnya sebagai bagian dari string dan bukan baris yang berisi string "50%" di dalam kolom Comments.

4. 5.6 Menggunakan Operator LIKE dengan ADO

Apabila Anda menjalankan pernyataan SQL pada database melalui *Active Data Ob*jects, Anda harus menggunakan sintaks SQL Server. ADO akan menangani simbol-simbol khusus saat menghubungkan ke database access. Pernyataan berikut ini tidak akan menampilkan baris-baris apapun jika Anda menjalankannya sebagai query Acces.

```
SELECT  CompanyName, ContactName, Country
```



```
FROM  Customers
WHERE  Country LIKE '_ermany'
```

Pernyataan yang tepat untuk Access adalah :

```
SELECT   CompanyName, ContactName, Country
FROM  Customers
WHERE   Country LIKE  '?ermany'
```

Tetapi pernyataan ini tidak dapat dijalankan pada SQL Server. Pernyataan yang dapat dijalankan pada SQL Server juga dapat dijalankan pada Access, jika Anda menjalankannya melalui ADO. Pernyataan pada pemrograman Visual Basic berikut ini akan mengambil 11 pelanggan dari Germany, baik dijalankan pada SQL Server atau Access.

```
cmd = "SELECT  CompanyName, ContctName, Country"
cmd = cmd "FROM Customers"
cmd = cmd & "WHERE Country LIKE '_ermany'
RS. Open Text1.Text, objConnection
```

Variabel *objConnection* mewakili objek Connection ke database Northwind.

Untuk informasi lebih lanjut mengenai cara menjalankan pernyataa SQL pada database sari dalam aplikasi VB, bacalah buku-buku pemrrograman database dengan VB.

4. 5.7 Mencari Nilai NULL

Sebuah operasi yang umum dalam memanipulasi dan mengelola database adalah mencari nilai-nilai Null di dalam field. Ekspresi IS NULL dan IS NOT FULL berfungsi mencari nilai-nilai field yang berupa NULL atau bukan NULL. Perbandingan string seperti yang berikut ini tidak akan mengembalikan nilai Null :

```
WHERE CompanyName = ” ”
```

Sebenarnya, pernyataan ini akan menghasilkan baris-baris pada kolom CompanyName yang berisi string kosong, tetapi bukan baris-baris yang tidak diberikan nilai. Baris-baris ini mungkin adalah baris yang diberikan nialai pada field CompanyName, tetapi nilainya dihapus kemudian hari. Mungkin ada alasan untuk mengisi nilai Null pada kolom CompanyName, gunakanlah klausa WHERE berikut ini :

```
Where CompanyName IS NULL
```


Harga NULL pada tabel Products berarti bahwa harga produk tidak tersedia pada saat pemasukan data. Anda bisa mencari produk tanpa harga dan menyuntingnya dengan mudah. Pernyataan berikut ini akan mencari produk tanpa harga :

```
WHERE Price IS NULL
```

Jika Anda membandingkan field price dengan nilai numerik nol, Anda akan menerima produk-produk dengan harga nol (ini bisa berarti produk yang tidak diproduksi lagi, produk gratisan, tetapi bukan produk yang tidak memiliki harga).

Berikut ini adalah sebuah contoh praktis. Sewaktu Anda mengetikkan judul buku pada database biblio, Anda mungkin tidak tahu nama penerbit buku tersebut dan mengosongkan filed ini. Di kemudian hari, Anda bisa mencari semua judul tanpa penerbit dan menyuntingnya kembali. Pernyataan berikut ini akan mengambil semua baris yang tidak mengandung nama penerbit.

```
SELECT Titles.Title
FROM  Titles
WHERE  Titles.PubID IS NULL
```

Pernyataan SQL yang lebih berguna dapat digunakan untuk mencari bukan hanya buku tanpa penerbit, tetapi juga juga buku-buku yang

menunjuk ke penerbit yang tidak ada. Sebagai contoh, Anda mungkin telah menghapus baris-baris pada tabel Publishers tanpa menghapus referensi yang berhubungan pada tabel Titles. Situasi ini tidak boleh muncul pada database yang dirancang dengan baik, tetapi kebanyakan database yang digunakan saat ini jauh dari sempurna, dan para konsultan menghadapi masalah yang serupa apabila mereka menangani proyek orang lain. Setelah Anda memahami pengertian dasar dari database, Anda harus memperhatikan aturan-aturan untuk menentukan integritas referensial. Sebagai contoh, database biblio tidak merepakan integritas referensial antara tabel titles dan Publishers. Dengan kata lain, Anda bisa menghapus nama penerbit walaupun ia ditunjuk oleh satu atau lebih judul buku.

Jika Anda menduga ada beberapa judul buku yang menunjuk ke penerbit yang tidak ada, gunakanlah pernyataan berikut ini untuk mengambil semua judul buku yang tidak memiliki penerbit atau merujuk ke penerbit yang tidak ada. Pernyataan ini akan mengambil judul buku tanpa penerbit :

```
SELECT  Titles.
FROM  Titles
WHERE (Titles.PubID IS NULL)
```


Pernyataan berikut ini akan mengambil judul buku dengan penerbit yang tidak sah :

```
SELECT  Titles.Title
FROM  Titles.Publisher
WHERE  Titles.PubID=[Publishers].[PubID] AND
        Publishers.Name IS NULL
```

Bagaimana kalau menggabungkan dua klausa WHERE dengan operator OR, agar Anda bisa mengambil semua judul buku dengan satu pernyataan? Pernyataan berikut ini kelihatnnya benar, tapi sebenarnya tidak :

```
SELECT  Titles.Title
FROM  Titles, Publishers
WHERE  ((Titles.PubID IS NULL) OR
(Titles.PubID=[Publishers].[PubID] AND
Publishers.Name IS NULL));
```

Pernyataan ini akan menampilkan banyak recordset yang mengandung semua kombinasi judul buku dan penerbit. Kita akan membahas mengenai query yang menghasilkan banyak recordset nanti dan membahasnya secara detail. Ada sebuah cara mudah untuk menggabungkan hasil dari dua atau lebih query dengan operator

UNION yang menggabungkan baris-baris unik dari dua recordset. Operator UNION akan dibahas pada bagian berikutnya.

Perhatikan bahwa anda tidak bisa menyunting tabel Titles untuk membuat judul-judul baru tanpa penerbit, karena field PubID pada tabel Titles tidak boleh berisi NULL. Anda harus menghapus pembatasan ini terlebih dahulu, lalu menyunting tabel tersebut. Setelah anda selesai, jangan lupa menambahkan pembatasan kembali ke dalam database.

4. 5.8 Operator UNION

Operator UNION berfungsi untuk menggabungkan hasil adri dua query yang terpisah ke dalam satu cursor. Cursor akhir merupakan gabungan dari cursor-cursor yang dihasilkan oleh masing-masing query, dan tidak mengandung baris-baris duplikat. Berikut ini adalah contoh dimana Anda bisa menggabungkan baris-baris yang diambil oleh dua contoh sebelumnya dengan operator UNION :

```
(SELECT Titles.Title
        FROM Titles,Publisher
        WHERE Titles.PubID IS NULL)
UNION
        (SELECT Titles.Title
                FROM Titles.Publishers
                WHERE Titles, PubID=Publishers.PubID AND
                Publishers.Name IS NULL)
```


Singkatnya, pernyatan ini akan mencari semua judul buku yang memiliki masalah pada entri penerbitnya. Ekspresi IS NULL dan IS NOT NULL digunakan sangat sering untuk menerapkan integritas database. Sebuah pernyataan SQL yang sederhana bisa mencari nilai Null di dalam tabel. Pada “pernyataan UPDATE”, Anda akan mempelajari cara menyunting field-field di dalam tabel dengan pernyataan SQL.

Bagaimana dengan record yang memiliki ID penerbit yang tidak sah (misalnya ID penerbit yang menunjuk ke record yang tidak ada pada tabel Publisher)? Untuk mencari baris-baris dari tabel Title yang menunjuk ke penerbit yang tidak ada, Anda harus menggunakan kata kunci IN (dan NOT IN).

4. 5.9 Kata Kunci DISTINCT

Kata kunci DISTINCT menghapus setiap duplikat dari cursor yang dihasilkan oleh pernyataan SELECT. Misalkan Anda ingin mencari negara mana yang memiliki pelanggan Northwind. Jika Anda mengambil semua nama negara dari tabel Customers, Anda akan mendapatkan banyak duplikat. Untuk menghapusnya, gunakanlah kata kunci DISTINCT, seperti ditunjukkan pada pernyataan berikukt ini :

```
SELECT  DISTINCT Country
FROM  Customers
```

Kata kunci DISTINCT diterapkan pada daftar field yang mengikutinya. Pernyataan berikut ini :

```
SELECT   DISTINCT
FROM   Customers
```

Akan mengambil nama negara dan kota yang unik. Pernyataan di atas mengambnil 70 buah baris. Jika Anda mengabaikan kata kunci DISTINCT, Anda akan mendapatkan 92 baris/ Ada tiga pelanggan pada Buenos Aires, Argentina, tetapi kombinasi kota-negara ini hanya akan muncul sekali bila pernyataan DISTINCT digunakan.

4. 5.10 Mengurutkan baris-baris dengan kata kunci ORDER

Baris-baris pada query jarang ditampilkan dalam urut-urutan yang sesuai dengan kehendak kita. Query diurutkan berdasarkan urut-urutan nama-nama kolom yang ditentukan pada pernyataan SELECT dan urut-urutan baris yang dimasukkan pada setiap tabel.Untuk menampilkan baris berdasarkan urut-urutan tertentu, gunakanlah klausa ORDER BY adalah

```
ORDER BY col1, col2…….
```


Anda bisa menentukan sebanyak mungkin kolom pada daftar Order. Output dari query diurutkan sesuai dengan nilai-nilai dari kolom pertama (col1). Jika ada dua baris yang memiliki nilai identik pada kolom ini, baris-baris tersebut akan diurutkan berdasarkan pada kolom kedua, dan seterusnya. Misalkan Anda ingin daftar semua pelanggan pada tabel Customers diurutkan berdasarkan negara.Karena Anda punya banyak pelanggan di setiap negara, Anda juga harus menentukan bagaimana setiap kelompok pelanggan dari negara yang sama akan diurutkan.

Pernyataan berikut ini :

```
SELECT   CompanyName, ContactName
FROM   Customers
ORDER  BY Country, City
```

Akan menampilkan pelanggan diurutkan berdasarkan negara dan berdasarkan kota di dalam setiap negara masing-masing. Jika Anda ingin mengurutkan baris-baris berdasarkan nama perusahaan di dalam setiap kelompok negara, gunakanlah pernyataan berikut ini :

```
SELECT   CompanyName, ContactName
FROM   Customers
ORDER  BY Country, CompanyName
```

Atau, Anda bisa mengurutkannya berdasarkan negara, berdasarkan kota di dalam setiap negara, dan berdasarkan nama perusahaan di dalam setiap kota dengan pernyataan berikut ini :

```
SELECT   Company Name, Contact Name
FROM  Customers
ORDER  BY  Country, City, CompanyName
```

4. 5.11 Field hasil Perhitungan

Selain nama-nama kolom, Anda bisa meletakkan field hasil perhitungan di dalam pernyataan SELECT. Tabel Order Details mengandung baris untuk setiap baris invoice. Sebagai contoh, invoice #1048 mengandung empat baris (empat item yang terjual), dan setiap baris perincian ditampilkan pada baris yang terpisah pada tabel Order Details. Setiap baris mengandung jumlah item yang terjual, harga item, dan diskon setiap barang. Untuk menmpilkan subtotal setiap baris, Anda harus mengalikan jumlah dengan harga dikurangi diskon, seperti ditunjukkan pada pernyataan berikut ini :

```
SELECT Orders.OrderID,Product ID.
        [Order Details].UnitPrice *
        [Order Details].Quantity * (1 – [Order  Details].Discount)
FROM Orders, [Order Details]
WHERE Orders. OrderID = [Order Details].OrderID
```


Pernyataan ini akan menghitung subtotal untuk setiap baris pada invoice yang dikeluarkan ke semua pelanggan Northwind dan menampilkannya bersamaan dengan nomor pesanan. Nomor pesanan diulang sebanyak jumlah produk di dalam pesanan (atau baris pada invoice).

PernyataanSQL terakhir terlalu panjang. Anda bisa memperpendeknya dengan mengabaikan nama tabel untuk field Quantity dan UnitPrice, karena nama-namanya tidak muncul pada tabel lain. Anda tidak bisa mengabaikan tabel dari nama field OrderID, karena ia muncul pada kedua tabel yang terlihat pada query. Anda juga bisa menentukan alias pendek untuk tabel Order Details dengan kata kunci Access lalu menggunakan alias ini sebagai pengganti nama tabel tersebut.

```
SELECT Orders.OrderID,UnitPrice * Quantity * (1-Discount)
FROM Orders, [Order Details] AS Details
WHERE Orders.OrderID = Details.OrderID
```

4. 5.12 Menjumlah dan Menghitung

SQL mendukung sejumlah fungsi yang dapat digunakan untuk menghitung field-field tertentu dari semua baris yang dihasilkan oleh query. Fungsi seperti ditampilkan pada tabel 4.2, melakukan kalkulasi dasar seperti menjumlahkan, menghitung, serta merata-rata nilai numerik. Fungsi-fungsi tersebut menerima anma-nama field (atau

field hasil perhitungan) sebagai argumen, dan menghasilkan satu nilai tunggal, yaitu nilao total (atau rata-rata) dari semua nilai.

Tabel 4.2 : Fungsi matematik SQL

Fungsi	Aksi
COUNT ()	Mengembalikan banyaknya nilai pada kolm yang ditentukan
SUM()	Mengembalikan jumlah nilai pada kolom yang ditentukan
AVG()	Mengembalikan nilai rata-rata pada kolom yang ditentukan
MIN()	Mengembalikan nilai terkecil pada kolom yang ditentukan
MAX()	Mengembalikan nilai terbesar pada kolom yang ditentukan

Fungsi-fungsi ini beroperasi pada satu kolom (yang mungkin bisa berupa kolom hasil perhitungan) dan mengembalikan satu nilai. Baris-baris yang terlibat di dalam perhitungan ditentukan dengan klausa WHERE yang tepat. Fungsi SUM() dan AVG() hanya bisa memproses nilai numerik. Ketiga fungsi yang lain bisa memproses baik nilai numerik dan teks.

Fungsi-fungsi ini digunakan untuk menyimpulkan data dari satu atau lebih tabel. Misalnya Anda ingin mengetahui berapa banyak pelanggan database Northwind yang terletak di Germany. Pernyataan SQL berikut ini akan mengembalikan nilai yang diinginkan :


```
USE NORTHWIND
SELECT COUNT(Customer ID)
FROMCustomers
WHERE Country = ‘Germany’
```

Ini adalah contoh sederhana dari fungsi COUNT(). Jika Anda ingin menghitung nilai unik, Anda harus menggunakan kata kunci DISTINCT bersamaan dengan nama dari field yang ingin dihitung. Jika Anda ingin mengetahui berapa banyaknya negara asal pelanggan Northwind, gunakanlah pernyataan SQL berikut ini :

```
SELECT COUNT(DISTINCT Country)
FROM Customers
```

Jika Anda mengabaikan kata kunci DISTINCT, pernyataan akan mengembalikan jumlah baris yang mengandung field Country. Pernyataan COUNT() mengabaikan nilai Null, kecuali jika Anda menentukan argumen *. Pernyataan berikut ini akan mengembalikan jumlah semua baris pada tabel Customers, bahkan jika beberapa diantaranya memiliki nilai Null pada kolom Country :

```
SELECT COUNT (*)
FROM Customers
```

Fungsi SUM() digunakan untuk menjumlahkan nilai pada field tertentu pada baris tertentu. Untuk mencari berapa banyaknya unit produk dengan ID = 11 (Queso Cabrales) yang telah terjual, gunakanlah pernyataan berikut ini :

```
USE NORTHWIND
SELECT SUM (Quantity)
FROM [Order Details]
WHERE ProductID=11
```

Pernyataan SQL yang mengembalikan pendapatan total tang dihasilkan oleh satu produk agak lebih rumit. Kali ini kita harus menjumlahkan kualitas produk dikalikan harga, serta mempertimbangkan pula diskon pada setiap invoice.

```
USE NORTHWIND
 SELECT SUM(Quantity)
FROM [Order Details]
WHERE ProductID=11
```

Anda bisa melihat kalau Queso Cabrales telah terjual 706 unit, yang menghasilkan pendapatan total sebesar $12,901.77.
Berikut ini adalah pernyataan SELECT yang menghasilkan semua ID produk bersamaan dengan nomor invoice yang mengandung ID

produk tersebut, serta kuantitas minimal, maksimal, dan rata-rata yang dipesan :

```
SELECT ProductID AS PRODUCT
        COUNT(ProductID) AS [INVOICE],
        MIN(Quantity) AS [MIN],
        MAX(Quantity) AS [MAX],
        AVG(Quantity) AS [Average]
FROM [Order Details]
GROUP BY ProductID
ORDER BY ProductID
```

4.5.12.1 Mengelompokkan Baris

Fungsi matematik beroperasi pada semua baris yang dipilih oleh query. Terkadang Anda perlu mengelompokkan hasil query, agar Anda bisa menghitung subtotal, misalkan Anda tidak hanya perlu menghitung pendapatan total yang dihasilkan oleh satu produk, tetapi daftar dari semua produk dan pendapatan yang dihasilkannya. Contoh terakhir pada bagaian sebelumnya menghitung pendapatan total yang dihasilkan oleh satu produk. Jika Anda mengabaikan klausa WHERE, ia akan menghitung pendapatan total yang dihasilkan oleh semua produk. Anda bisa menggunakan fungsi SUM() untuk memenggal perhitungan pada setiap ID produk baru seperti ditunjukkan query berikut. (Untuk melakukannya, Anda harus mengelompokkan ID produk bersama dengan klausa GROUP BY)

```
USE NORTHWIND
SELECT ProductID, SUM(Quantity * UnitPrice * (1-Discount))
                AS [Total Revenues]
FROM [Order Details]
GROUP BY ProductID
ORDER BY ProductID
```

Pernyataan di atas akan menghasilkan output seperti berikut ini :

ProductID	Total Revenues
1	12788.10
2	16355.96
3	3044.0
4	8567.89
5	5347.20
6	7137.0
7	22044.29
8	12772.0
9	7226.5
10	20867.34
11	12901.77
12	12257.66

CATATAN Anda akan melihat semua 77 buah ID produk, tetapi pada contoh di atas hanya ditulis sebagian.

Seperti yang bisa Anda lihat, fungsi SUM() bekerja sama dengan klausa GROUP BY (jika ada) untuk menghasilkan subtotal. Klausa GROUP BY bukanlah sebuah mekanisme pengaturan seperti klausa ORDER BY. Fungsinya mengelompokkan semua baris dengan nilai yang sama pada kolom yang ditentukan dan memerintahkan fungsi untuk menghitung setiap grup secara terpisah. SQL akan mengurutkan baris-baris sesuai dengan kolomyang ditentukan pada klausa GROUP BY dan mulai menghitung fungsi. Setiap kali menemukan grup baru, ia akan mencetak hasilnya dan me-reset fungsi.

Jika Anda menggunakan klausa GROUP BY pada pernyataan SQL, Anda harus mengerti aturan berikut ini : *Semua field yang disertakan pada daftar SELECT harus merupakan bagian dari fungsi atau bagian dari klausa GROUP BY*. Misalnya Anda ingin mengubah pernyataan sebelumnya untuk menampilkan nama-nama produk, bukan ID produk. Pernyataan berikut ini akan menampilkan nama produk, bukan ID produk. Perhatikan bahwa field ProductName tidak muncul sebagai argumen fungsi, jadi ia harus menjadi bagian dari klausa GROUP BY.

```
USE NORTHWIND
SELECT ProdukName,
                SUM (Quantity * [Order Details].UnitPrice *
                (1-discount)) AS [TOTAL Revenues]
FROM [Order Details], Products
WHERE  Products.Product ID =[Order Details], ProductID
GROUP  BY  ProductName
ORDER  BY  ProductName
```

Berikut ini adalah beberapa baris pertama dari output yang dihasilkan oleh pernyataan diatas :

ProductName	Total Revenues
Alice Mutton	32698.38
Anissed Syrup	3044.0
Boston Crab Meat	17910.63
Camember Pierrot	46927.48
Carnavron Tigers	29171.87
Chai	12788.10
Chan	16355.96
Chartreuse Verte	12294.54
Chef Anton's Cajun Seasoning	8567.89
Chef Anton's Gumbo Mix	5347.20
Chocolate	1368.71
Cote de blaye	141396.74

4.5.12.2 Cara Menggunakan Fungsi Matematik

Fungsi Matematik hanaya bisa digunakan pada daftar field yang mengikuti pernyataan SELECT, Atau pada klausa HAVING. Fungsi tidak bisa digunakan pada klausa FROM atau WHERE. Selain itu, jika

daftar SELECT mengandung fungsi matematik, Anda tidak bisa menyertakan filed-field lain pada daftar yang sama, kecuali jika mereka bagian dari fungsi matematik, atau bagian dari klausa GROUP BY. Pernyataan berikut ini :

```
SELECT ProductID.
        SUM (Quantity * UnitPRice * (1-Discount)
        AS REVENUE
FROM [Order Details]
WHERE ProductID=11
```

Tidak menampilkan ID produk bersama dengan pendapatan yang dihasilkan oleh produk tertentu. Tetapi akan menghasilkan pesan error berikut ini :

> *Column 'Order Details.ProductID' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause.*

Jika Anda mencoba menjalankan pernyataan yang sama sebagai query Access, pesan error yang akan muncul adalah :

> *You tried to execute a query that doesn't include the specified expression 'ProductID' as part of an aggregate function.*

Untuk menghindari pesan error ini dan menampilkan ID produk serta pendapatan yang dihasilkannya, gunakanlah klausa GROUP BY seperti pada pernyataan berikut ini. Anda sebenarnya tidak perlu mengelompokkan satu produk, tetapi itu dapat dilakukan dan Anda akan mendapatkan output yang diinginkan.

```
SELECT ProductID
        SUM(Quantity * UnitPrice * (1-Discount)
        AS REVENUE
FROM [Order Details]
WHERE ProductID=11
GROUP BY ProductID
```

Pernyataan SELECT berikut ini akan menghasilkan semua ID produk serta nomor invoice yang mengandungnya, serta kuantitas minimal, maksimal, dan rata-rata yang dipesan :

```
SELECT ProductID AS PRODUCT
        COUNT (ProductID) AS [INVOICES],
        MIN (Quantity) AS [MIN],
        MAX (Quantity) AS [MAX],
        AVG (Quantity) AS [AVERAGE]
FROM [Order Details]
GROUP BY ProductID
ORDER BY ProductID
```


4.5.12.3 Membatasi Grup dengan HAVING

Klausa HAVING membatasi grup-grup yang akan muncul pada cursor. Klausa ini mirip dengan WHERE, yang membatasi jumlah baris yang akan muncul pada cursor hasil. Tetapi klausa HAVING hanya bisa digunakan pada klausa GROUP BY, dan semua field yang digunakan pada klausa HAVING juga harus muncul pada daftar GROUP BY, atau sebagai argumen pada fungsi matematik.

Pernyataan berikut ini akan menghasilkan ID produk yang penjualannya melebihi 1000 unit :

```
USE NORTHWIND
SELECT ProductID, SUM(Quantity)
FROM [Order Details]
GROUP BY ProductID
HAVING SUM (Quantity)>1000
```

Jika Anda melihat nama produk dan bukan ID Produk, Anda harus menambahkan pernyataan yang agak lebih panjang yang mengandung tabel Products dan memetakannya ke dalam ProductID pada tabel Order Details dengan klausa WHERE :

```
USE NORTHWIND
SELECT Prodcts.ProductName, [Order Details].ProductID,
```

```
        SUM (Quantity) AS [Items Sold]
FROM Products, [Order Details]
WHERE [Order Details]. ProductID = products,ProductID
GROUP BY [Order Detalis].ProductID,Products.ProductName
HAVING SUM (Quantity)>1000
ORDER BY Products.ProductName
```

Gunakanlah klausa WHERE untuk menyertakan pembatasan tambahan. Untuk mengambil semua produk “mahal” dengan jumlah penjualan yang besar, gantilah klausa WHERE pada contoh sebelumnya dengan pernyataan berikut ini :

```
WHERE [Order Details}.ProductID = Products.ProductID AND
        Products.UnitPrice > = 50
```

(Kita misalkan produk mahal adalah produk-produk yang harganya $50 atau lebih.) Metode yang lebih baik untuk mencari produk=produk mahal adalah dengan menghitung harga rata-rata dan menggunakannya untuk memilih produk yang harganya lebih tinggi dari harga produk rata-rata.

4. 5.13 Kata Kunci IN dan NOT IN

Kata kunci IN dan NOT IN digunakan pada klausa WHERE untuk menentukan daftar nilai yang harus sesuai (atau tidak sesuai) dengan kolom tertentu. Kedua klausa ini lebih seperti notasi singkat untuk

banyak operator OR. Berikut ini adalah sebuah pernyataan sederhana yang mengambil nama-nama pelanggan di Germany, Austria, dan Italy (semuanya 16 baris) :

```
USE NORTHWIND
SELECT CompanyName
FROM Customers
WHERE Country =’Germany’ OR
        Country=’Auatria’OR
        Country=’Italy’
```

Bayangkan jika kita ingin memilih pesanan yang mengandung satu dari banyak produk, atau pelanggan dari semua negara di eropa. Pernyataan yang samaharus ditulis ulang dengan bantuan kata Kunci IN seperti berikut ini :

```
USE NORTHWIND
SELECT CompanyName
FROM Customers
WHERE Country IN (‘Germany’, ‘Austria’, Italy’)
```

Pernyataan yang kedua ini lebih singkat, namun lebih mudah dibaca dan diedit.

4. 5.14 Subquery

Kata Kunci IN dan NOT IN juga bisa digunakan dengan subqueery. Subquery adalah sebuah query biasa yang cursor hasilnya digunakan sebagai bagian dari query lain. Misalnya Anda ingin mencari produk yang tidak pernah dipesan oleh orang Austria. Langkah pertama adalah membuat daftar produk yang pernah dipesan oleh perusahaan di Austria. Berikut ini adalah pernyataan yang mengembalikan ID dari produk-produk ini :

```
USE NORTHWIND
SELECT DISTINCT ProductID
FROM [Order Details], Customers, Orders
WHERE [Order Details].OrderID = OrderID AND
WHERE [Orders.CustomerID = Customers.CustomerID AND
Customers.Country = “Asutria”
```

Pernyataan ini mengembalikan daftar ID produk yang telah dijual kepada perusahaan-perusahaan di Austria. Kita ingin ID produk yang tidak muncul pada daftar ini, dan kita bisa mendapatkannya dengan pernyataan lain yang menggunakan operator NOT IN untuk mencari ID yang todak terdapat pada daftra yang dihasilkan oleh query di atas. Query ini akan menjadi subquery untuk pernyataan berikut Ini :

```
USE NORTHWIND
SELECT Product ID, ProductName
```



```
FROM Products
WHERE ProductID NOT IN
        (SELECT DISTINCT ProductID
        FROM [Order Details]. Customers, Orders
        WHERE [Order.CustomerID = Customers.CustomerID AND
        Customers.Country = “Austria”
```

Seluruh pernyataan SQL yang mengambil produk yang dijual di Austria ditutupi oleh sepasang tanda kurung yang menjadi daftara dari oparator NOT IN.

Anda mungkin berpikir adakah cara yang lebih sederhana. Jika Anda mengubah tanda sama dengan pada subquery menjadi tidak sama denagn, Anda akan mendapatkan produk-produk yang tidak pernah dijual ke Austria. Jika Anda mengubah :

```
Customers.Country = “Austria”
```

Menjadi

```
Customers.Country < > “Austria”
```

Anda akan mendapatkan produk-produk yang telah dijual ke semua negara kecuali Austria. Ini adalah hasil yang berbeda debnagn produk-produk yang tidak dijual ke Austria. Kemungkinannya bahwa semua

produk pada database telah dijual ke setidaknya negara lain, apakah itu dijual ke pelanggan Austria atau tidak.

4. 5.15 Kata Kunci BETWEEN

BETWEEN berfungsi menentukan suatu kisaran nilai dan membatasi pemilihan baris yang hanya mengandung kolom tertentu sesuai kisaran nilai tersebut. BETWEEN adalah notasi singkat dari ekspresi berikut ini :

```
Column >= minValue AND column <= maValue
```

Untuk mengambil pesanan yang dilakukanb pada kuartal pertama 1997, gunakanlah pernyataan berikut ini :

```
USE NORTHWIND
SELECT OrderID, OrderDate, CompanyName
FROM Orders, Customers
WHERE Orders.CustomerID = Customers.CustomerID AND
        (OrderDate BETWEEN '1/1/1997' AND '3/31/1997')
```

Daftar pesanan mengandung pesanan-pesanan yang dilakukan pada tanggal pertama dan tanggal terakhir dari interval yang ditentukan.

Query Access yang mengandung tanggal harus dibatasi dengan tanda#. Query Access yang setara dengan di atas adalah sebagai berikut :

```
USE Northwind
SELECT OrderID, OrderDate, CompanyName
FROM Orders, Customers
WHERE Orders, CustomerID = Customers.CustomerID AND
        (OrderDate BETWEEN #1/1/1997# AND #3/31/1997#)
```

Anda bisa menggunakan operator BETWEEN untuk menentukan kisaran nilai numerik (seperti harga) dan string (dari A sampai E, misalnya). Pernyataan berikut ini :

```
USE     Northwind
SELECT CompanyName
FROM Customers
WHERE CompnayName BETWEEN 'A' AND  'E'
```

Akan mengambil semua pelanggan dengan nama perusahaan yang diawali dengan huruf A, B, C, D, dan E (jika ada). Perusahaan dengan nama seperti Eastern Connection secara alfabetis lebih besar dari E dan tidak akan disertakan pada cursor. Untuk menyertakan nama perusahaan yang namanya diawali dengan E, gunakanlah pernyataan berikut ini :

```
USE       Northwind
SELECT CompanyName
FROM Customers
WHERE CompnayName BETWEEN ‘A’ AND  ‘EZZZ’
```

Pernyataan terakhir bisa ditulis ulang dengan operator LIKE :

```
SELECT CompanyName
FROM Customers
WHERE CompnayName LIKE ‘[A-E]%’
```

4. 5.16 Menggabungkan Tabel-tabel

Klausa WHERE mengizinkan Anda menggabungkan dua atau lebih tabel, berdasarkan pada nilai dari kedua kolom di dalam kedua tabel. Tetapi klausa WHERE harus digunakan untuk mengekspresikan batasan yang melibatkan satu atau lebih tabel, dan bukan untuk menghubungkan tabel (walaupun banyak programmer yang menggunakan klausa WHERE untuk itu). Metode yang tepat untuk menghubungkan tabel adalah dengan operasi JOIN. Sebelum kita melihat JOIN, mari kita bahas tentang kelemahan WHERE untuk menghubungkan banyak tabel. Dua kolom yang dilibatkan pada klausa WHERE biasanya merupakan kunci primer dari sebuah tabel dan kunci asing pada tabel lain. Apabila Anda menghubungkan tabel

Titles ke tabel Publisher (Publisher.PubID) sama dengan kunci asing pada tabel titles (field Titles.PubID).

```
WHERE Titles.PubID = Publisher.PubID
```

Kolom-kolom yang Anda yang Anda gunakan untuk menggabungkan kedua tabel tidak perlu diindeks. Jika Anda mengabaikan klausa WHERE, query akan menghasilkan hasil perkalian dengan baris-baris pada kedua tabel (yaitu semua kombinasi yang mungkin dari baris-baris pada kedua tabel).

Menggabungkan banyak tabel dengan klausaWHERE tidak dapat dilakukan bila salah satu dari kedua kolom mengandung nilai NULL. Pada contoh ini, kolom Publishers.PubID tidak bisa memiliki nilai Null, tetapi Titles.PubID bisa. Judul buku dengan nilai Null pada kolom PubID tidak akan cocok dengan baris apapun pada tabel Publishers, dan ia akan dilewatkan. Anda mungkin tidak membutuhkan baris-baris ini, tetapi biasanya kita membutuhkan semua judul buku, baik buku itu memiliki penerbit atau tidak. Ini berlaku juga untuk pengarang./ Query Allbooks juga mengambil seluruh judul buku dengan pengarang dari database Biblio. Tetapi ia akan menolak judul buku tanpa pengarang. Anda bisa memeriksa ini

dengan melihat judul buku “Database Processing” yang tidak memiliki pengarang. Judul ini tidak akan disertakan pada query.

Mari kita perhatikan kembali query Allbooks yang diulangi di sini :

```
SELECT Titles.Title, Publishers.Name, Authors.Author
FROM Titles, [Title Author], Authors, Publishers
WHERE Titles.PubID = Publisher.PubID AND
        Titles.ISBN = [Title Author].ISBN AND
        [TITLE Author].Au_ID = Authors.Au_ID
ORDER BY Titles.Title
```

Query ini mengembalikan 15.959 baris saat dijalankan pada komputer saya. Karena ada sebanyak 8531 judul buku pada database Biblio, dan kebanyakan dari judul buku-buku ini memeiliki dua atau tiga pengarang. Sewaktu saya mengambil judul buku dan pengarangnya dengan menggunakan JOIN, hasil query adalah sebanyak 17.449 baris. Perbedaannya sebesar 17.449 – 15.959 = 1490 baris adalah jumlah buku yang tidak memiliki pengarang. (Jika Anda telah menyunting database Biblio, Anda akan mendapatkan jumlah yang berbeda, tetapi Anda bisa memeriksa perbedaan ini).

Untuk membuktikan kesimpulan ini, kita harus menghitung jumlah buku tanpa pengarang langsung dari database. Untuk melakukanya, kita akan menghitungjudul buku yang ISBN-nya tidak muncul pada

tabel Title Author. Pertama-tama, kita harus harus mengambil semua nilai ISBN dari tabel Title Author. Query ini cukup sederhana :

```
SELECT DISTINCT ISBN FROM [Title Author]
```

Lalu Anda harus mengambil semua judul buku yang ISBN-nya tidak muncul pada hasil query di atas. Lebih baik lagi, Anda tinggal menghitung baris-baris dari tabel Titles, dimana ISBN-nya tidak terdapat pada hasil query yang pertama :

```
SELECT COUNT (Title)
FROM Titles
WHERE Titles.ISBN NOT IN
        (SELECT DISTINCT ISBN FROM [Title Author])
```

Jika Anda menjalankan pernyataan ini pada database Biblio (gunakanlah database Access jika Anda belum mengubahnya ke dalam database SQL Server), ini akan membuktikan kebenaran penghitungan kita (ia akan menampilkan sebanyak 1.490 judul buku).

Pertanyaannya sekarang adalah, bagaimana kita bisa mengambil semua judul buku, baik buku tersebut memiliki pengarang atau tidak. Operasi penggabungan banyak tabel disebut dengan istilah JOIN, dan bisa dilakukan dengan operasi JOIN. Operasi JOIN dirancang secara

khusus untuk menggabungkan banyak tabel. Klausa WHERE mampu melakukan hal ini pada situasi yang sederhana, tetapi tidak selalu berhasil, seperti ditunjukkan pada contoh sebelumnya.

Operasi JOIN menggabungkan kolom-kolom dari kedua tabel dimana baris-barisnya memiliki field yang sesuai. Sintaksnya adalah :

```
FROM Table1 INNER JOIN Table1.co1 = Table2.co1
```

CATATAN Anda bisa mengabaikan kata kunci INNER untuk saat ini. Anda akan melihat nanti bahwa ada berbagai jenis penggabungan, database Anda harus menentukan jenis penggabungan yang ingin dilakukan dengan kata kunci lain.

Kedua kolom tidak perlu dicocokkan dengan operator sama dengan, walaupun metode ini paling umum digunakan untuk emncocokkan banyak tabel. Anda bisa menggujnakan salah satu dari eperator relasional (>, >=, <, <= dan <>). Operator JOIN mungkin ahnya muncul pada pernyataan SELECT pada klausa FROM, dan itulah sebabnya saya menyertaklan kata kunci FROM pada sintaks pernyataan tersebu. Sebagai tambahan, Anda bisa menggabungkan banyak pembatasan dengan opartor logikal. Sebagai contoh, Anda

mungkin ingin ada sepasang kolom yang benar-benar cocok, dan kolom lain berbeda :

```
FROM Table1 INNER JOIN table2 ON Table.col1 = Table2.col2.col1
AND
        Table1.col2 <> Table2.col2
```

Pernyataan ini akan mengambil semua baris dimana kolom col1 pada Table sesuai dengan col2 pada table2 dan tidak menyertakan baris-baris dimana nilai col2 berbeda.

4. 5.17 Jenis-jenis Penggabungan

SQL mendukung dua jenis penggabungan :

1. **Inner Join** Penggabungan ini menghasilkan semua pasangan baris-baris yang cocok pada kedua tabel dan membuang baris-baris yang tidak cocok dari kedu atabel. Ini adalah setting default jika Anda tidak menentukan jenis penggabungan.
2. **Full Join (atau Outer Join)** Penggabungan ini menghasilkan semua baris yang biasanya dihasilkan oleh operasi INNER JOIN ditambah baris-baris dari tabel kiri atau kanan yang tidak memenuhi kondisi penggabungan. Full join akan menghasilkan semua judul buku yang memiliki penerbit serta judul buku tanpa penerbit. Baris-baris dengan kolom-kolom yang tidak cocok pada setiap tabel diset menjadi NULL. Pada contoh judul

buku/penerbit, judul buku tanpa penerbit akan muncul dengan nilai Null pada kolom penerbit.

Baik inner dan outer join bisa dimodifikasi dengan menentukan tabel mana yang akan digunakan sebagai basis dari penggabungan. Sebagai contoh, kita mungkin ingin mengambil semua baris dari satu tabel dan mencocokkan baris-baris pada tabel lainnya. Oleh karena itu, penggabungan bisa lebih diklasifiaksikan lagi sesuai dengan urut-urutan mereka pada tabel-tabel yang disertakan pada operasi :

1. **Left Join** penggabungan inner join ini menghasilkan semua record dari tabel yang pertama (kiri) adri kedua tabel, bahkan jika tidak ada nilai yang cocok pada record pada tabel yang kedua (kanan)
2. **Right Join** Penggabungan inner join ini menghasilkan semua record dari tabel yang kedua (kanan) dari kedua tabel, bahkan jika tidak ada nilai yang cocok pada record pada tabel yang pertama (kiri).

Untuk mendemostrsikan jenis-jenis penggabungan yang berbeda, saya telah membuat sebuah database yang rumit dengan dua tabel : sebuah tabel dengan nama kota (Cities) dan sebuah tabel dengan nama negara (Countries). Setiap negara memiliki ID yang unik, yang digunakan oleh tabel Cities untuk menemukan di negara mana sebuah kota

terdapat. Field CountryID merupakan kunci primer pada tabel Countries dan kunci asing pada tabel Cities. Berikut ini adalah struktur dan kedua tabel dan isinya :

Tabel Cities

City Name	varchar (50)
CountryID	int

Tabel Countries

CountryName	varchar (50)
CountryID	int

Tabel diatas menunjukkan semua entri pada kedua tabel di dalam database dan pemetannya. Jika sebuah kota tidak dipetakan pada sebuah negara, maka kolom kedua memiliki nilai Null pada baris kota tersebut.Jika sebuah negara tidak dipetakan pada sebuah kota, maka kolom pertama memiliki nilai Null pada baris negara tersebut.

CityName	CountryName
NULL	Argentina
NULL	China
NULL	France
NULL	Japan
Alexandria	Egypt
Barcelona	NULL
Berlin	Germany

Cairo	Egypt
Chicago	USA
Frankfurt	Germany
Hamburg	Germany
London	UK
Madrid	NULL
Manchester	UK
New York	USA
Rome	Italy
San Fransisco	USA
Venice	Italy
Zurich	NULL

Seperti yang Anda bisalihat, tabel Cities mengandung kota-kota yang tidak terdapat pada negara manapun (Barcelona, Madrid, dan Zurich), dan tabel Countries mengandung nama-nama negara yang tidak dirujuk oleh baris manapaun pada tabel Cities (Japan, France, Argentina, dan China). Mari kita gabungkan kedua tabel ini dengan menggunakan semua variasi JOIN.

INNER JOIN

Query pertama menggunakan INNER JOIN, yang mengambil baris-baris yang sesuai saja :

```
USE CitiesDB
SELECT Cities.CityName, Countries.CountryName
FROM Cities INNER JOIN Countries
```



```
ON Cities,CountryID = Countries.CountryID
ORDER BY Countries.CountryName, Cities.CityName ;
```

Hasil dari query ini ditunjukkan di Bawah ini. Dua belas baris kota dipetakan ke negara. Kota-kota yang tidak memiliki negara tidak disertakan pada hasil ini.

CityName	CountryName
Alexandria	Egypt
Cairo	Egypt
Berlin	Germany
Frankfurt	Germany
Hamburg	Germany
Rome	Italy
Venice	Italy
London	UK
Manchester	UK
Chicago	USA
New York	USA
San Fransisco	USA

LEFT JOIN

Jika Anda mengganti INNER JOIN dengan LEFT JOIN pada pernyataan sebelumnya, query akan mengambil semua kota, baik ia memiliki negara atau tidak. Jika sebuah kota tidak dipetakan kepada negara, baris yang bersesuaian pada kolom kedua akan mengandung nilai NULL. Query ini akan menghasilkan 15 baris (jumlah baris pada

tabel kiri). Pernyataan yang mengambil semua kota adalah sebagai berikut :

```
USE CitiesDB
SELECT Cities.CityName, Countries.CountryName
FROM Cities LEFT JOIN Countries
        ON Cities.CountryID = Countries.CountryID
ORDER BY Countries.CountryName, Cities.CityName
```

Berikut ini hasil dari query :

City Name	CountryName
Barcelona	NULL
Madrid	NULL
Zurich	NULL
Alexandria	Egypt
Cairo	Egypt
Berlin	Germany
Frankfurt	Germany
Hamburg	Germany
Rome	Italy
Venice	Italy
London	UK
Manchester	UK
Chicago	USA
New York	USA
San Fransisco	USA

RIGHT JOIN

Jika Anda menggunakan RIGHT JOIN, query akan mengambil semua negara, baik negara tersebut dipetakan ke kota atau tidak. Jika ada sebuah negara yang tidak dipetakan pada kota, baris yang bersesuaian pada kolom kedua akan mengandung nilai Null. Query ini akan menghasilkan 16 baris (jumlah baris pada tabel kanan).

Pernyataan yang mengambil semua negara adalah sebagai berikut :

```
USE CitiesDB
SELECT Cities.CityName, Countries.CountryName
FROM Cities RIGHT JOIN Countries
        ON Cities.CountryID = Countries.CountryID
ORDER BY Countries.CountryName, Cities.CityName
```

Berikut ini adalah hasil dari query :

City Name	CountryName
NULL	Argentina
NULL	China
Alexandria	Egypt
Cairo	Egypt
NULL	France
Berlin	Germany
Frankfurt	Germany
Hamburg	Germany
Rome	Italy
Venice	Italy

NULL	Japan
London	UK
Manchester	UK
Chicago	USA
New York	USA
San Fransisco	USA

FULL JOIN

Kata kunci FULL JOIN mengembalikan semua baris dari kedua tabel. Semua kota memiliki nilai negara (Walaupun itu nilai Null), dan semua negara memiliki nilai kota (Walaupun itu nilai Null). Nama-nama negara akan diulangi sesuai kebutuhan. Hasilnya tidak mengandung duplikat nama kota, karena sebuah kota tidak bisa terdapat pada lebih dari satu negara.

```
USE CitiesDB
SELECT Cities.CityName, Countries.CountryName
FROM Cities FULL JOIN Countries
        ON Cities.CountryID = Countries.CountryID
ORDER BY Countries.CountryName, Cities.CityName
```

Hasil dari query ini mengandung 19 baris, sebagai berikut :

CityName	CountryName
NULL	Argentina
NULL	China

NULL	France
NULL	Japan
Alexandria	Egypt
Barcelona	NULL
Berlin	Germany
Cairo	Egypt
Chicago	USA
Frankfurt	Germany
Hamburg	Germany
London	UK
Madrid	NULL
Manchester	UK
New York	USA
Rome	Italy
San Fransisco	USA
Venice	Italy
Zurich	NULL

Kata kunci FULL JOIN tidak didukung oleh Access dan Anda tidak bisa menguji pernyataan SQL yang terakhir tanpa SQL Server (atau DMB lain yang mendukung full join). Tetapi full join tidak sering digunakan, dan bisa menghasilkan cursor yang cukup besar.

Semoga database sederhana ini bisa membantu Anda memahami konsep inner dan outer join. Kini kita bisa mengguanakan kata kunci ini untuk menulis beberapa query tingkat lanjut yang menggabungkan banyak tabel dengan oprator JOIN.

4.6 Action Query

Selain selection query, SQL mendukung beberapa pernyataan untuk action query. Kita menggunakan action query untuk memanipulasi isi dari database. Anda bisa melakukan tiga jenis tindakan : penyisipan, pembaruan, dan penghapusan. SQL menyediakan tiga pernyataan untuk melakukan ini : *INSERT, UPDATE* dan *DELETE*. Sintaksnya sederhana, jadi Anda tidak perlu mempelajari perintah tambahan lagi. Seperti yang bisa Anda tebak, tindakan ini tidak bisa dilakukan pada semua baris pada tabel. Baris-baris yang akan dipengaruhi oleh action query ditentukan oleh klausa WHERE yang tepat. Sebagian besar informasi yang Dibahas pada bagian sebelumnya juga bisa diterapkan pada action query.

Perbedaan yang utama antara selection query dan action query adalah bahwa action query tidak menghasilkan baris-baris. Apabila Anda menjalankan action query dengan Query Analayzer, Anda akan melihat jumlah baris yang dipengaruhi pada panel Result. Access memiliki pendekatan yang berbeda. Ia menjalankan action query, tetapi tidak langsung meng-update tabel. Pertama-tama sebuah kotak pesan memberi tahu user berapa berapa banyak baris yang akan diubah oleh query. Pada titik ini Anda bisa melakukan perubahan atau membatalkan action query.

Aplikasi SQLEXEC tidak melaporkan jumlah baris yang akan diubah, tetapi menampilkan jumlah baris yang telah dipengaruhi oleh action query pada baris judulnya.

4.6.1 Pernyataan DELETE

Mari kita mula dengan action query yang paling sederhana, yang berungsi menghapus satu atau lebih baris pada tabel. Pernyataan DELETE berfungsi menghapus seluruh baris, dan Anda tidak perlu menentukan daftar field. Tetapi Anda harus menentukan baris mana yang harus dihapus. Sintaks dari pernyataan DELETE adalah sebagai berikut :

```
DELETE FROM table_name
WHERE condition
```

Dimana table_name adalah nama dari tabel dimana baris-barisnya akan dihapus. Hanya baris yang memenuhi kondisi yang ditentukan oleh klausa WHERE yang akan dihapus. Pernyataan

```
DELETE FROM table_name
WHERE Country = ‘Germany’
```

Akan menghapus semua pelanggan dari Germany dari database Northwind. Klausa WHERE dari pernyataan DELETE (serta pernyataan lainnya) bisa menjadi lebih rumit. Anda juga bida mengabaikannya, dan dalam hal ini semua baris akan dihapus dari tabel. Dalam kasus ini, Anda bahkan tidak perlu menggunakan kata kunci FROM. Anda tinggal menentukan nama tabel dengan pernyataan seperti berikut ini :

```
DELETE  CATEGORIES
```

Pernyataan ini akan menghapus semua baris pada tabel Categories. Tetapi pernyataan DELETE dipengaruhi oleh aturan integritas referensial yang diterapkan ke dalam database. Baris-baris pada tabel Categories dirujuk oleh baris-baris pada tabel Products, dan DMBS (Access atau SQL Server) akan menolak untuk menghapusnya. Jika ada kategori yang tidak dirujuk oleh produk manapun, ia akan dihapus.

Pernyataan tidak bersyarat akan menghapus semua baris pada tabel yang tidak dirujuk sebagai kunci asing pada tabel lain. Jika Anda menggunakan SQL Server, ada cara yang lebih baik yaitu dengan menggunakan pernyataan TRUNCATE TABLE, yang berfungsi menghapus baris-baris yang sama hanya lebih cepat. Untuk

menghapus semua kategori yang tidak digunakan sebagai kunci asing pada tabel Products, gunakanlah pernyataan berikut ini :

```
TRUNCATE  TABLE Categories
```

4.6.2 Pernyataan INSERT

Pernyataan INSERTY berfungsi menyisipkan sebuah baris baru ke dalam tabel. Sintaks yang lengkap dari pernyataan ini adalah :

```
INSERT table_name (column list) VALUES (value list)
```

Pernyataan ini akan menambahkan sebuah baris ke dalam tabel table_name dan menugaskan nilai yang ditentukan ke dalam field-field baris. Kedua daftar harus ditutup dengan tanda kurung, seperti ditunjukkan di atas. Nilai pertama dari value list diberikan pada kolom pertama pada column list, nilai kedua pada value list akan diberikan pada kolom kedua pada column list, dan seterusnya. Sudah jelas kedua daftar ini harus sesuai satu sama lain. Keduanya harus memiliki jumlah item yang sama, serta jenis nilainya harus sesuai denagn jenis kolom. Field identiras tidak bisa ditentukan pada column list. DBMS akan memberikan nilai kepad field-field ini secara otomatis.

Untuk menyisipkan pelanggan baru ke dalam tabel Customers, gunakanlah pernyataan berikut ini :

```
INSERT (CustomerID, CompanyName, ContactName)
    Values (“SYBEX”, “Sybex, Inc.” “Tobias Smythe”)
```

Pernyataan ini tidak memberikan nilai ke semua kolom pada baris baru. Hanya tiga dari kolom tersebut yang akan memiliki nilai. Anda bisa meng-update baris dengan pernyataan UPDATE kapan saja (dibahas berikutnya). Kolom-kolom yang tidak ditampilkan pada column list akan berisi default (jika ada) atau nilai Null. Jika kolom tidak memiliki nilai default dan tidak boleh berisi nilai Null, Anda harus memberikan nilai ke dalamnya.

Jika Anda ingin memebrikan nilai untuk setiap field pada baris baru, Anda bisa mengabaikan column list, selama Anda memasukkannya dalam urutan yang tepat.

Cara lain untuk memberikan nilai kolom ke dalam pernyataan INSERT adalah dengan menggunakan SELECT dari tabel lain. Misalnya Anda ingin menyisipkan alamat kontak dari tabel Customers ke dalam tabel PhoneBook. Langkah pertama adalah mengambil informasi yang dibutuhkan dari tabel Customers :

```
SELECT ContactName, Phone, Fax
FROM Customers
```

Daftar ini lalu bisa menjadi value list dari pernyataan INSERT :


```
INSERT PhoneBook
   SELECT ContactName, Phone, Fax
   FROM Customers
```

Bentuk pernyataan INSERT ini bisa digunakan untuk menyisipkan banyak baris sekaligus.

4.6.3 Pernyataan UPDATE

Pernyataan action query yang terakhir adalah UPDATE, yang berfungsi meng-update kolom-kolom pada baris tertentu. Pernyataan UPDATE bisa mempengaruhi banyak baris, tetapi tetap berada pada satu tabel. Sintaksnya adalah :

```
UPDATE table_name
   SET column1 = value1, column2 = value2,…
WHERE condition
```

Pernyataan berikut ini akan mengubah kolom Country dari baris-baris pada tabel Customers yang berisikan pelanggan dari Germany.

```
UPDATE Customers
SET Country = ‘Germany’
WHERE Country = ‘W. Germany’ OR Country = ‘E. Germany’
```

Berikut ini adalah pernyataan yang jauh lebih rumit yang meng-update harga produk tertentu. Misalnya pemasok barang telah menaikkan diskon sebesar 5% dan Anda ingin memberikan separuh dari diskon ini kepada pelanggan Anda. Pernyataan berikut ini akan menyesuaikan harga produk yang Anda beli dari pemasok dengan ID=4 sebesar 2.5 %. Berikut ini adalah pernyataan UPDATE :

```
UPDATE Products
SET UnitPrice = UnitPrice * 0.025
WHERE SupplierID = 3
```

BAB V

APLIKASI SQL

5.1 Buku dengan Banyak Pengarang

Contoh berikut ini menggunakan database Biblio dimana kita akan mencari buku dengan banyak pengarang dengan menggunakan fungsi COUNT () untuk menghitung jumlah pengarang pada setiap buku. Bagaimana caranya mengubah pernyataan seperti “untuk setiap buku” ke dalam pernyataan SQL? Mudah, Anda hanya perlu mengelompokkan hasil berdasarkan ISBN (Anda tidak bisa menggunakan judul buku, karena mungkin ada buku yang memiliki judul yang sama). Karena Anda tidak mempedulikan nama pengarang, tetapi hanya jumlahnya, Anda tidak akan menggunakan tabel Authors. Jumlah pengarang bisa diambil dari tabel Title Author.

```
SELECT Titles.Title, COUNT([Title Author].Au_ID
FROM [Title Author], TITLES
WHERE Titles.ISBN = [Title Author].ISBN
```

Pernyataan ini tidak bisa dijalankan, karena kedua field pertama pada daftar SELECT tidak muncul sebagai fungsi atau klausa GROUP. Untuk menjadikannya pernyataan SQL yang sah, kita harus menambahkan klausa GROUP Byseperti ditunjukkan di sini.


```
SELECT Titles.Title, Count([Title Author].Au_ID
FROM [Title Author], Titles
WHERE Titles.ISBN = [Title Author].[ISBN]
GROUP BY Titles.Title, [Title Author].ISBN
```

Jika Anda menjalankan query ini pada database Biblio, Anda akan mendapatkan daftar buku-buku pada tabel titles bersamaan dengan jumlah pengarang, tambahkanlah klausa HAVING berikut ini ke dalam pernyataan sebelumnya :

```
HAVING COUNT ([Title Author}.ISBN) > 1
```

5.2 Pengarang dengan Banyak Buku

Contoh berikut ini tidak jauh beda dari yang sebelumnya. Kali ini kita akan mencari pengarang yang telah menulis lebih dari satu buku. Kita ingin menampilkan nama-nama pengarang dan jumlah buku yang telah mereka tulis. Kita tidak ingin menampilkan judul buku yang mereka tulis, jadi kita tidak akan menggunakan tabel title pada query ini. Pernyataan SELECT mengandung sebuah nama field akan fungsi COUNT :

```
SELECT Authors.Author, Count ([Title Author].ISBN) AS Books
FROM [Title Author], Authors
```

Baris-baris dari kedua tabel harus memiliki field yagng sama, yaitu ISBN dari buku, Sebagai tambahan, field Authors.Author harus muncul pada klausa GROUP BY. Berikut ini adalah pernyataan SQL yang lengkap:

```
SELECT Authors.Author, Count ([Title Author].ISBN) AS Books
FROM [Title Author], Authors
WHERE Authors.Au_ID=[Title Author].[Au_ID]
GROUP BY Authors.Author
```

Sekali lagi, jika Anda ingin membatasi agar yang ditampilkan hanya penulis yang menulis lebih dari satu buku saja, gunakanlah klausa HAVING :

```
HAVING COUNT ([Title Author].ISBN) > 1
```

5.3 Pelanggan, Pesanan, Perincian

Contoh terakhir dari bagian ini melibatkan empat tabel yang terdapat pada database Northwind. Kita akan membuat sebuah pernyataan yang mengambil semua pelanggan, pesanan mereka, serta perincian setiap pesanan pada satu laporan. Output yang diinginkan ditunjukkan berikut ini. Perhatikan bahwa saya telah menyingkat nama pelanggan dan produk, agar informasi dapat muat pada halaman ini. Cursor yang

sesungguhnya mengandung lebih banyak baris daripada yang ditunjukkan di sini :

Company	OrderID	Product	Price	Qty	Disc	ExtPrice
Alfreds	10643	Rossle	45.60	15	25	513.00
Alfreds	10643	Chartreuse	18.00	21	25	283.50
Alfreds	10643	Spegesild	12.00	2	25	18.00
Alfreds	10692	Vegie	43.90	20	0	878.00
Alfreds	10702	Aniseed	10.00	6	0	60.00
Alfreds	10702	Lakkalikoori	18.00	15	0	270.00
Alfreds	10835	Raclette	55.00	15	0	825.00
Alfreds	10835	Original	13.00	2	20	20.80
Alfreds	10952	Grandma’s	25.00	16	5	380.00
Alfreds	10952	Rossle	45.60	2	0	91.20
Alfreds	11011	Escargots	13.25	40	5	503.50
Alfreds	11011	Flotemysost	21.50	20	0	30.00
Ana	10308	Gudbrandsdal	28.80	1	0	28.80
Ana	10308	Outback Lager	12.00	5	0	60.00
Ana	10625	Tofu	23.25	3	0	69.75
Ana	10625	Singaporean	14.00	5	0	70.00
Ana	10625	Camembert	34.00	10	0	340.00
Ana	10759	Mascarpone	32.00	10	0	320.00
Ana	10926	Questo	21.00	2	0	42.00
Ana	10926	Konbu	6.00	10	0	60.00
Ana	10926	Teatime	9.20	7	0	64.40
Ana	10926	Mozarella	34.80	10	0	348.00

Seperti biasa, kita akan mulai dengan daftar kolom-kolom yang diinginkan :

```
SELECT CompanyName, OrderID, ProductName,
        [Order Details].UnitPrice AS Price,
        Quantity. Discount * 100 AS Discount,
        Quantity * (1-Discount) *
        [Order Details].UnitPrice AS Subtotal
FROM Products, [Order Details], Customers, Orders
```

Field-field yang diinginkan adalah nama perusahaan (CompanyName), ID invoice (OrderID), dan setiap perincian invoice : nama produk, harga, kuantitas, diskon, dan harga total. Harga total adalah sebuah field hasil perhitungan, yang dihitung dengan rumus sebagai berikut :

```
Quantity * Price * (1-Discount)
```

Berikutnya, kita harus menyediakan batasan untuk mencocokkan field-field yang dipilih dalam keenam tabel. Jika tidak, hasil perkalian yang dihasilkan oleh query ini akan salah. Batasan tersebut adalah sebagai berikut :

- Setiap ID produk pada tabel Oredr Details harus sesuai dengan baris pada tabel Products


```
[Order Details]. ProductID = Products.ProductID
```

- Setiap ID produk pada tabel Order Details harus sesuai dengan baris pada tabel Orders :

```
[Order Details].OrderID = Orders.OrderID
```

- Setiap ID pelanggan pada tabel Orders harus sesuai dengan baris pada tabel Customers :

```
Orders.CustomerID = Customers.CustomerID
```

Yang terakhir, kita harus mengurutkan baris-baris cursor berdasarkan CompanyName dan OrderID, agar setiap invoice pelanggan muncul bersama-sama di dalam bagian pelanggan. Berikut ini adalah pernyataan SQL yang lengkap yang mengambil pelanggan, pesanan, serta perincian pesanan dari database Northwind :

```
USE NORTHWIND
SELECT CompanyName, Orders. OrderID, ProductName,
        [Order Details].UnitPrice AS Price,
        Quantity. Discount * 100 AS Discount,
        Quantity * (1-Discount) *
        [Order Details].UnitPrice AS [Ext. Price]
FROM Products, [Order Details], Customers, Orders
WHERE [Order Details].ProductID = Products.ProductID AND
```

```
[Order Details].ProductID = Orders.OrderID AND
Orders.CustomerID=Customers.CustomerID
ORDER BY Customers.CompanyName, Orders.orderID
```

5.4 Penerbit, Buku, Pengarang

Mari kita akhiri pembahasan ini dengan pernyataan SQL yang serupa yang berfungsi mengambil semua judul buku pada database Biblio serta mengurutkannya nerdasarkan penerbitnya. Di dalam setiap penerbit, buku-buku juga diurutkan berdasarkan penerbitnya. Di dalam setiap penerbit, buku-buku juga diurutkan berdasarkan judul dan pengarangnya. Format yang diinginkan dari outputnya adalah sebagai berikut :

Company Name	Title	ISBN	Author
AK PETERS LTD	Aphysical Approach…..	1-5688101-3-X	Levitus
AK PETERS LTD	Colour Principles….	1-56881009-1	Cowan
AK PETERS LTD	Colour Principles….	1-56881009-1	Shumaker
AK PETERS LTD	Computer Facial …	1-5688101-4-8	Parke
AK PETERS LTD	Computer Facial ….	1-5688101-4-8	Werner
AK PETERS LTD	Computer Graphics…	1-5688100-8-3	Brown
AK PETERS LTD	Computer Graphics…	1-5688100-8-3	Pressman
AK PETERS LTD	Fundamentals of….	1-5688100-7-5	Hampe
AK PETERS LTD	Fundamentals of….	1-5688100-7-5	Hoschek
AK PETERS LTD	Fundamentals of….	1-5688100-7-5	Lasser
AK PETERS LTD	Fundamentals of….	1-5688100-7-5	Schumak
AK PETERS LTD	Language for….	0-8672045-0-8	Ayer
AK PETERS LTD	Language for…	0-8672045-0-8	Myers
…			

SYBEX Inc	Microsoft Access….	0-7821176-5-1	Reddick
SYBEX Inc	Microsoft Access….	0-7821176-2-7	Getz
SYBEX Inc	Microsoft Access….	0-7821176-2-7	Powell
SYBEX Inc	Microsoft Access….	0-7821121-3-7	Briggs
SYBEX Inc	Microsoft Access….	0-7821121-3-7	Powell

Mari kita kembali ke contoh kita dan menghasilkan daftar seperti yang ditunjukkan di atas, dengan pernyataan dan perintah-perintah yang telah kita pelajari sejauh ini. Pernyataan ini harus memilih field-field sebagai berikut :

```
SELECT Titles.Title,Publisher.Name,Authors.Author
    FROM Titles, [Title Author], Author, Publishers
```

Ini adalah bagian yang paling sederhana dari pernyataan SQL kita. Kini Anda harus membangun klausa WHERE yang menghubungkan judul buku dengan pengarang dan penerbitnya. Klausa untuk menggabungkan judul buku dan penerbit cukup sederhana :

```
WHERE Titles.PubID = Publishers.PubID
```

Untuk menghubungkan judul buku dan pengarang, Anda harus mempertimbangkan tabel Title Author, yang berada dia antara tabel Titles dan tabel Authors.

```
WHERE Titles.ISBN = [Title Author].ISBN AND
      [Title Author].Au_ID = Authors.Au_ID
```

Pernyataan yang lengkap yang menggabungkan ketiga pembatasan di atas ditunjukkan di bawah. Saya telah menambahkan klausa ORDER BY untuk mengurutkan judul buku secara alfabetis.

```
SELECT Titles.Title, Publishers.Name, Authors.Author
FROM Titles, [Title Author], Authors, Publishers
WHERE Titles.PubID = Publisher.PubID AND
      Titles.ISBN = [Title Author].ISBN AND
      [TITLE Author].Au_ID = Authors.Au_ID
ORDER BY Titles.Title
```

5.5 Menggabungkan judul buku dan Pengarang

Mari kita menginmplementasikannya dengan JOIN, karena ingin menampilkan semua judul buku dengan pengarangnya, Anda harus menggunakan dua left join. Pertama-tama, Anda harus menggabungkan tabel Title Author dan Authors dengan operasi LEFT JOIN. Hasilnya adalah sebuah tabel virtual dengan semua ISBN di sebelah kiri dan nama-nama pengarang yang sesuai di sebelah kanan. Bahkan ISBN yang muncul pada tabel Title Author tetapi tidak

memiliki pengarang yang sesuai pada tabel Author s akan ditampilkan pada hasilnya. Tabel virtual ini kemudian harus digabungkan dengan table Titles, dengan operasi LEFT JOIN yang lain. Tabel Titles akan berada di sisi kiri dari penggabungan pertama, agar judul-judul buku yang tidak memiliki entri pada tabel Title Author akan disertakan pada hasilnya. Listing 4.9 menunjukkan query yang mengambil semua judul buku dan pengarangnya dengan operasi JOIN :

```
SELECT Titles.Title, Authors. Author
FROM Titles
      LEFT JOIN ([Title Author])
            LEFT JOIN Authors
            ON [Title Author].Au_ID = Authors.Au_ID)
                  ON Titles.ISBN =[Title Author].ISBN
ORDER BY Titles.Title;
```

Pernyataan ini diproses sebagai berikut : SQL Server di mulai dengan mengevaluasi penggabungan paling dalam, yang menggabungkan baris-baris pada tabel Title Author dan Authors. Penggabungan ini memilih semua baris pada tabel Title Author dan kolom pada tabel Authors yang memiliki nilai Au_ID yang sama. Jika baris pada tabel kiri tidak memiliki baris yang sesuai pada tabel kanan, ia tetap akan disertakan pada hasil query. Jika baris pada tabel kanan tidak memiliki baris yang sesuai pada tabel kiri, ia akan diabaikan. (Ini berarti nama

pengarang yang nerujuk ke sebuah judul buku telah dihapus dari tabel Titles, tetapi entrinya pada tabel Title Author dan Authors belum dihapus). Berikut ini adalah penggabungan pertama yang akan dijalankan oleh DBMS :

```
([Title Author] LEFT JOIN Authors
ON [Title Author].Au_ID = Authors.Au_ID)
```

Hasil dari penggabungan ini berupa sebuah tabel lain (tabel ini hanya akan muncul selama eksekusi query). Kita sebut saja tabel ini ISBNAuthors. Jika Anda mengganti ekspresi dari penggabungan paling dalam dengan ISBNAuthors, query aslinya akan menjadi :

```
SELECT Titles.Title, Authors. Author
FROM Titles
        LEFT JOIN ISBNAuthors
                ON Titles.ISBN =[Title Author].ISBN
ORDER BY Titles.Title;
```

Setelah penciptaan penggabungan paling dalam ini, DBMS akan menjalankan penggabungan lainnya. Kali ini ia akan menggabungkan baris-baris pada tabel Titles dengan baris-baris yang dihasilkan oleh penggabungan yang telah dijalankan dengan mencocokkan ISBN.

5.6 Penerbit, Buku, Pengarang

Contoh berikutnya mengembangkan contoh sebelumnya dengan menyertakan tabel Publisher. Kali ini kita akan menambahkan sebuah penggabungan ketiga untuk menghubungkan judul buku dan penerbit. Tabel perantara ini lalu akan menjadi tabel pertama (kiri) dari penggabungan lainnya yang menghasilkan judul buku, penerbit, dan pengarang.

```
SELECT Publishers.[Company Name], Titles.Title, Authors.Author
FROM (Titles LEFT JOIN Publisher
        On Titles.PubID = Publishers.PubID)
        LEFT JOIN ([Title Author] LEFT JOIN Authors
                ON [Titles Author].Au_ID = Authors.Au_ID) ON
        Titles.ISBN = [Title Author].ISBN
ORDER BY Publishers.[Company Name], Titles.Title
```

5.7 Pelanggan, Pesanan, Perincian

Contoh ini akan mengambil informasi yang berhubungan dengan penjualan dari database Northwind. Kita ingin mengambil semua pelanggan. Pesanan, serta perincian pesanan pada query dengan struktur berikut ini :

CustomerName	OredrID	ProdName	QTY
	Price	Disc.	Subtot
Alfreds Futt.	10643	Rossle	15
	45.60	0.25	513.0
Alfreds Futt.	10643	Chartreuse	21
	18.00	0.25	283.5
Alfreds Futt.	10643	Spegesild	2
	12.00	0.25	18.0
Alfreds Futt.	10692	Vegie	20
	43.90	0.0	878.0
Alfreds Futt.	10702	Aniseed	6
	10.00	0.0	60.0
Alfreds Futt.	10702	Lakkalik	15
	18.00	0.0	270.0

Mari kita membuat pernyataan SELECT yang mengambil informasi ini. Pertama-tama, kita harus memilih semua pelanggan. Di dalam setiap bagian pelanggan, kita harus menampilkan semua pesanannya (ID serta tanggal pesanan dilakukan). Yang terakhir, di dalam setiap bagian pesanan, kita ingin menampilkan semua perincian pesanan. Field-field ini didapatkan dari tabel Customers, Orders, dan Order Details. Perhatikan bahwa tabel Order Details mengandung ID produk dan bukan nama produk yang sesungguhnya, jadi kita harus menyertakan tabel products. Untuk setiap ID produk pada tabel Order Details, kita harus mengambil nama produk dari baris yang sesuai pada tabel Products.

Mari kita mulai dengan tabel Customers dan menentukan CompanyName sebagai field pertama pada daftar SELECT. Tabel Customers harus digabungkan dengan tabel Orders pada filed CustomersID. Penggabungan ini akan mengulangi setiap nama pelanggan sebanyak mungkin selama ada pesanan untuk pelanggan tersebut. Output dari penggabungan pertama ini adalah sebagai berikut:

Alfreds Futterkiste	10643
Alfreds Futterkiste	10692
Alfreds Futterkiste	10702
Alfreds Futterkiste	10835
Alfreds Futterkiste	10952
Alfreds Futterkiste	11011
Ana Trujillo Emparedados y helados	10308
Ana Trujillo Emparedados y helados	10625
Ana Trujillo Emparedados y helados	10759
Ana Trujillo Emparedados y helados	10926

(Pelanggan pertama memiliki enam pesanan, pelanggan kedua memiliki tiga pesanan, dan seterusnya.) Berikut ini adalah penggabungan yang menghasilkan daftar di atas :

```
SELECT Customers.CompanyName,Orders.OrderID
FROM   Customers INNER JOIN Orders
```

```
                On Customers,CustomerID = Orders.CustomerID
    ORDER BY Customers.CompanyName
```

Sekarang kuta harus mengambil tabel Order Details dan menggabungkannya dengan tabel Orders pada field OrderID. Jika pesanan pertama memiliki tiga item, baris

```
Alfreds Futterkiste        10643
```

Harus diulangi sebanyak tiga kali. Mari kita tingkatkan pernyataan terakhir untuk menyertakan tabel Order Details. Untuk tujuan contoh ini, kita akan menggunakan kolom UnitPrice, Quantity, dan Discount dari tabel Order Details :

```
SELECT Customers.CompanyName, Orders.OrderID,
  [Order Details].ProductID, [Order Details].Quantity,
  [Order Details].UnitPrice
FROM   (Customers INNER JOIN Orders
  ON Customers.CustomerID = Orders.CustomerID)
        INNER JOIN [Order Details]
        ON [Order Details].OrderID=Orders.OrderID
ORDER BY Customers.CompanyName
```


Perhatikan penempatan tanda kurung pada klausa FROM. Tanda kurung ini tidak dibutuhkan, tetapi saya ingin menekankan bahwa penggabungan pertama (yang telah kita rancang pada langkah pertama) mempengaruhi tanbel lain. Ini disebut dengan tabel virtual, karena tidak terdapat di dalam database, tetapi SQL Server harus membuatnya untuk menggabungkan baris-barisnya dengan tabel Order Details.

Kini kita harus mencocokkan field Order Details.ProductID dengan field ProductID dari tabel Products, agar kita menampilkan nama produk yang sesuangguhnya. Berikut ini adalah pernyataan SQL yang terakhir :

```
SELECT Customers.CompanyName, Orders.OrderID,
  Products.ProductName,[Order Details].Quantity,
      [Order Details].UnitPrice,[Order Details].Discount,
          [Order Details]. Quantity * [Order
          Details].UnitPrice * (1-[Order
          Details].Discount) AS Subtotal
FROM   (Customers INNER JOIN Orders
  ON Customers.CustomerID = Orders.CustomerID)
      INNER JOIN [Order Details] ON
          [Order Details].OrderID=Orders.OrderID
      INNER JOIN Products ON
          [Order Details].ProductID = Products.ProductID
ORDER BY Customers.CompanyName
```

Saya telah menghapus field ProductID dari daftar SELECT dan menggantinya dengan field Products.ProductName. Selain itu, saya telah menambahkan sebuah field hasil perhitungan yang merupakan hasil subtotal (quantity * (price – discount)).

Berikut ini adalah pernyataan SQL yang mengambil informasi yang sama, tetapi juga menghitung total berjalan :

```
SELECT Customers.CompanyName, Orders.OrderID,
   Products.ProductName
   [Order Details].Quantity, [Order Details].UnitPrice
   [Order Details]. Quantity * (1 - Discount) *
   [Order Details].UnitPrice AS Subtotal
FROM   Customers, Orders, [Order Details], Products
WHERE Orders. CustomerID = Customers.CustomerID AND
   [Order Details].OrderID=Orders.OrderID AND
   [Order Details].ProductID = Products.ProductID
ORDER BY Customers.CompanyName, Order.OrderID
COMPUTE SUM(Quantity * (1 – Discount) * [Order Details].Unit
Price)
        BY Customers.CompanyName, Orders.OrderID
COMPUTE      SUM(Quantity * (1 – Discount) * [Order
Details].Unit Price)
        BY Customers.CompanyName
```


CATATAN Pernyataan COMPUTE hanya dikenali oleh SQL

Query Engine dari SQL Server (atau Access) dioptimalkan untuk penggabungan, yang merupakan operasi yang akan sangat sering Anda jalankan pada database Anda. Untuk membantu Query Engine, Anda harus mempertimbangkan mengindeks kolom-kolom yang dilibatkan pada query yang paling umum digunakan.