BASIS DATA

BUKU AJAR

Raissa Amanda Putri, S.Kom.,M.TI.

Buku Ajar

BASIS DATA

Raissa Amanda Putri, S.Kom.,M.TI.

Buku Ajar
BASIS DATA
Edisi Pertama

Perpustakaan Nasional: Katalog Dalam Terbitan (KDT)
ISBN 978-623-91257-7-6
15 x 23 cm
110 hlm
Cetakan ke-1, Desember 2021

Penulis
Raissa Amanda Putri, S.Kom.,M.TI.

Editor
Dr. Mhd. Furqan, S.Si.,M.Comp.Sc.

Desain Sampul
Ulfayani Mayasari, M.Si.

Tata Letak
Franindya Purwaningtyas, MA.

Penerbit
Al Azhar Centre
Ikatan Alumni Universitas Al-Azhar Mesir
Jl. Jalan Jangka, Gang Roda, No.69, Medan
Telp : 082362218683
Email: azharcentre.publishing@gmail.com

KATA PENGANTAR

Bismillahirrahmanirrahim…

Alhamdulillah, puji dan syukur kepada Allah SWT karena atas rahmat dan karuniaNya maka penulis dapat menyelesaikan Buku Ajar Mata Kuliah Basis Data ini.

Penulis mengucapkan terima kasih atas dukungan dan bantuan dari orang tua, suami, anak-anak, para pimpinan, rekan – rekan dosen, dan teman sejawat di lingkungan Universitas Islam Negeri Sumatera Utara Medan atas terselesaikannya buku ini. Semoga Buku Ajar Mata Kuliah Basis Data ini dapat bermanfaat serta membantu dan mendukung tercapainya tujuan dari proses belajar mengajar terutama di lingkungan UIN Sumatera Utara Medan.

Penulis menyadari bahwa masih banyak kekurangan dan keterbatasan pada buku ini, dan penulis mengharapkan kritik dan saran dari berbagai pihak agar buku ini dapat diperbaiki pada revisi berikutnya. Akhir kata, semoga segala upaya yang penulis lakukan ini dapat bermanfaat bagi kita semua dan mendapat berkah dari Allah SWT. Aamiin…

Medan, Desember 2021

Raissa Amanda Putri, S.Kom.,M.T.I.

SILABUS

DESKRIPSI MATA KULIAH

Mata kuliah Basis Data merupakan mata kuliah yang memberikan konsep dasar basis data kepada mahasiswa, bagaimana membuat model data, merancang basis data dengan baik, normalisasi basis data serta menjelaskan sistem informasi dimana basis data dapat diterapkan.

TUJUAN MATA KULIAH

Setelah mengikuti mata kuliah ini diharapkan mahasiswa dapat merancang, membuat, dan mengelola basis data, baik yang berbasis komputer tunggal atau basis data berbasis web, serta mampu mengaplikasikannya dalam dunia kerja.

CAPAIAN PEMBELAJARAN

1. Mahasiswa mampu menjelaskan konsep dasar basis data.
2. Mahasiswa mampu merancang, membuat dan mengelola basis data.
3. Mahasiswa mampu mengaplikasikan basis data ke dalam sistem.

MATERI PEMBELAJARAN

1. Pengantar Basis Data
2. Basis Data Relasional
3. Structured Query Language (SQL)
4. Entity Relationship (ER)
5. Perancangan Basis Data
6. Normalisasi
7. Query

DAFTAR ISI

KATA PENGANTAR i
SILABUS ii
DAFTAR ISI iii
DAFTAR GAMBAR vi
DAFTAR TABEL vii
BAB I 1
Pengantar Basis Data 1
1.1. Basic File System 1
1.2. Basis Data 3
1.3. Database Management System (DBMS) 4
1.4. Aplikasi Sistem Basis Data 7
1.5. Tujuan Sistem Basis Data 9
1.6. Bahasa Basis Data 16
1.7. Latihan 18
BAB II 20
Basis Data Relasional 20
2.1. Model Data 20
2.2. Struktur Basis Data Relasional 21
2.3. Skema Basis Data 23
2.4. Kunci 25
2.5. Skema Diagram 27
2.6. Latihan 28
BAB III 29
Structured Query Language (SQL) 29
3.1. SQL Data Definition Language (DDL) 29
3.1.1. Tipe Dasar Variabel SQL 29
3.1.2. Definisi Skema Dasar 31
3.2. Kueri pada Satu Tabel 36
3.3. Kueri pada Multi Tabel 40

3.4. Natural Join 43
3.5. Operasi Dasar Tambahan 46
3.5.1. Operasi *Rename* 46
3.5.2. Operasi *String* 48
3.5.3. Spesifikasi Atribut pada Klausa *Select* 50
3.5.4. Menyusun Tampilan *Record* 51
3.5.5. Predikan Klausa *Where* 52
3.6. Operasi *Set* 52
3.6.1. Union 54
3.6.2. Intersect 57
3.6.3. Except 58
3.7. Null 60
3.8. Modifikasi Basis Data 62
3.8.1. Delete 62
3.8.2. Insert 64
3.8.3. Update 66
3.9. Latihan 67
BAB IV 69
Entity Relationship (ER) 69
4.1. Entity Relationship Model 69
4.2. Entitas 69
4.2.1. Entitas Kuat dan Entitas Lemah .73
4.2.2. Entitas Asosiatif 73
4.3. Atribut 74
4.3.1. Atribut Sederhana dan Komposit 75
4.3.2. Atribut Bernilai Tunggal dan Multinilai 75
4.3.3. Atribut yang diturunkan 76
4.4. Relasi 77
4.5. Kardinalitas 81
4.6. Entity Relationship Diagram (ERD) .83
4.6.1. Struktur Dasar ERD 84

4.6.2. Derajat Relasi 85
4.6.3. Batasan Kardinalitas 88
4.7. Latihan 90
BAB V 91
Perancangan Basis Data..... 91
5.1. Masalah Penyimpanan Data..... 91
5.2. Proses Perancangan Basis Data 93
5.3. Latihan 95
BAB VI..... 96
Normalisasi..... 96
6.1. Manfaat Normalisasi..... 97
6.2. Ketergantungan dalam Normalisasi.. 97
6.3. Tahapan Normalisasi 99
6.4. Latihan 105
BAB VII 106
Query 106
7.1. Membuat Basis Data 106
7.2. Membuat Tabel 106
7.3. Menambah Data 107
7.4. Mengubah Data..... 108
7.5. Menghapus Data 108
7.6. Latihan 108
DAFTAR PUSTAKA..... 109

DAFTAR GAMBAR

Gambar 2.1. Skema Diagram untuk Basis Data Universitas 28
Gambar 4.1. Set Relasi Penasehat 78
Gambar 4.2. Pemetaan Kardinalitas (a)satu-ke-satu (b)satu-ke-banyak 83
Gambar 4.3. Pemetaan Kardinalitas (a)banyak-ke-satu (b)banyak-ke-banyak 83
Gambar 4.4. ERD Dosen dan Mahasiswa 84
Gambar 4.5. Derajat Relasi (a)Unary (b)Binary (c)Ternary 85
Gambar 4.6. *Unary* (a)satu-ke-satu (b)satu-ke-banyak (c)banyak-ke-banyak 86
Gambar 4.7. *Binary* (a)satu-ke-satu (b)satu-ke-banyak (c)banyak-ke-banyak 87
Gambar 4.8. *Ternary* 88
Gambar 4.9. Relasi dengan batasan kardinalitas 89
Gambar 4.10. Kardinalitas Relasi (a)Mandatory One (b)Mandatory Many (c)Optional One (d)Optional Many 90

DAFTAR TABEL

Tabel 2.1. Tabel Dosen 22
Tabel 2.2. Tabel Mata Kuliah 22
Tabel 2.3. Tabel Prasyarat 23
Tabel 2.4. Tabel Jurusan 24
Tabel 3.1. Hasil Pencarian Nama Dosen 37
Tabel 3.2. Hasil Pencarian Jurusan Dosen 37
Tabel 3.3. Hasil Pencarian Jurusan Dosen dengan *distinct* 38
Tabel 3.4. Hasil Kueri Multi Tabel 41
Tabel 3.5. Mata Kuliah Semester 1 tahun 2017 53
Tabel 3.6. Mata Kuliah Semester 2 tahun 2018 53
Tabel 3.7. Tabel Hasil *union* 54
Tabel 3.8. Tabel Hasil *union all* 55
Tabel 3.9. Tabel Hasil *intersect* 57
Tabel 3.10. Tabel Hasil *intersect all* 58
Tabel 3.11. Tabel Hasil *except* 59
Tabel 3.12. Perbandingan Nilai *Unknown* 60
Tabel 4.1. Set Entitas Dosen 72
Tabel 4.2. Set Entitas Mahasiswa 72
Tabel 4.3. Simbol Dasar ERD 84
Tabel 5.1. Contoh Tabel Redundansi 92
Tabel 6.1. Tabel Mentah 99
Tabel 6.2. Tabel *Unnormalized* 100
Tabel 6.3. Tabel Normal 1 (1NF) 101
Tabel 6.4. Functional Dependency 102
Tabel 6.5. Tabel Mahasiswa 2NF 102
Tabel 6.6. Tabel Mata Kuliah 2NF 103
Tabel 6.7. Tabel Nilai 2NF 103
Tabel 6.8. Tabel Mahasiswa 3NF 104
Tabel 6.9. Tabel Dosen 3NF 104

Tabel 6.10. Tabel Mata Kuliah 3NF 105
Tabel 6.11. Tabel Nilai 3NF 105

BAB I

Pengantar Basis Data

1.1. Basic File System

Sistem basis data merupakan hasil evolusi dari sistem *file*. Sistem *file* menyimpan data selama periode waktu yang lama, dan memungkinkan penyimpanan sejumlah besar data. Namun, sistem *file* umumnya tidak menjamin bahwa data tidak dapat hilang jika tidak didukung, dan tidak mendukung akses efisien ke item data yang lokasinya di *file* tertentu yang tidak diketahui.

Selanjutnya, sistem *file* tidak secara langsung mendukung bahasa permintaan untuk data dalam *file*. Dukungan untuk skema terbatas pada pembuatan struktur direktori untuk *file*. Kemungkinan pengguna dapat kehilangan data yang belum dicadangkan. Akhirnya, sistem *file* tidak memuaskan. Selain itu, sistem *file* memungkinkan akses bersamaan ke *file* oleh beberapa pengguna atau proses, tetapi sistem *file* umumnya tidak akan mencegah situasi seperti dua pengguna memodifikasi *file* yang sama pada waktu yang hampir bersamaan, sehingga perubahan yang dilakukan oleh satu pengguna gagal muncul di *file*.

Perusahaan memiliki banyak koleksi data tentang karyawan, departemen, produk, penjualan, dan sebagainya. Data ini diakses secara bersamaan oleh beberapa karyawan. Permintaan data harus dijawab dengan cepat,

perubahan yang dilakukan pada data oleh pengguna yang berbeda harus diterapkan secara konsisten, dan akses ke bagian data tertentu seperti data gaji dan data privasi lainnya harus dibatasi.

Mengelola data dengan menyimpannya dalam *file* sistem operasi memiliki banyak kelemahan, antara lain:

1. Perusahaan mungkin tidak memiliki memori utama yang cukup besar untuk menampung semua data. Karena itu data harus disimpan ke dalam perangkat penyimpanan seperti disk dan membawa bagian-bagian yang relevan ke dalam memori utama untuk diproses sesuai kebutuhan.
2. Bahkan jika memiliki memori utama yang besar, pada sistem komputer dengan pengalamatan 32-bit, tidak dapat merujuk secara langsung ke lebih dari sekitar 4 GB data. Harus ada program khusus yang terdiri dari beberapa metode untuk mengidentifikasi semua item data.
3. Harus ada program khusus untuk menjawab setiap permintaan yang mungkin ingin ditanyakan pengguna tentang data tersebut. Program-program ini cenderung rumit karena volume besar data yang akan dicari.
4. Data harus dilindungi dari perubahan yang tidak konsisten yang dilakukan oleh pengguna yang berbeda yang mengakses data secara bersamaan. Jika aplikasi harus membahas perincian akses bersamaan tersebut, ini menambah kerumitannya.

5. Harus dipastikan bahwa data dikembalikan ke keadaan konsisten jika sistem macet saat perubahan sedang dilakukan.
6. Sistem operasi hanya menyediakan mekanisme kata sandi untuk keamanan. Ini tidak cukup fleksibel untuk menegakkan kebijakan keamanan di mana pengguna yang berbeda memiliki izin untuk mengakses bagian data yang berbeda.

Database Management System (DBMS) adalah perangkat lunak yang dirancang untuk membuat tugas-tugas sebelumnya lebih mudah. Dengan menyimpan data dalam DBMS alih-alih sebagai kumpulan *file* sistem operasi, fitur-fitur DBMS dapat digunakan untuk mengelola data dengan cara yang kuat dan efisien. Ketika volume data dan jumlah pengguna menumbuhkan ratusan *gigabyte* data dan ribuan pengguna adalah hal yang biasa dalam database perusahaan, saat ini, dukungan DBMS menjadi sangat diperlukan.[1]

1.2. Basis Data

Data adalah fakta – fakta yang menggambarkan suatu kejadian yang sebenarnya pada waktu tertentu. Data didapatkan dari suatu kejadian yang benar – benar terjadi, misalnya dari transaksi penjualan, pembelian, dan sebagainya. Data identik dengan bukti transaksi yang terjadi di suatu perusahaan seperti kwitansi, faktur, formulir dan lain – lain. Data yang telah diproses kemudian dapat menghasilkan informasi berupa

[1] Christopher J Date, "An Introduction to Database Systems 8th Edition," 2004.

laporan, seperti laporan keuangan, laporan penjualan, dan sebagainya.

Basis data merupakan kumpulan informasi yang ada selama periode waktu yang lama, seringkali bertahun-tahun. Basis data merupakan hal yang sangat penting untuk semua bisnis. Basis data berada di belakang layar perusahaan besar maupun perusahaan kecil. Perusahaan menyimpan setiap data – data penting mereka ke dalam basis data. Kekuatan basis data berasal dari pengetahuan dan teknologi yang telah berkembang dan diwujudkan dalam perangkat lunak khusus yang disebut sistem manajemen basis data, atau *Database Management System* (DBMS), atau disebut juga Sistem Basis Data.

1.3. Database Management System (DBMS)

DBMS adalah alat yang ampuh untuk membuat dan mengelola jumlah data yang besar secara efisien dan memungkinkannya bertahan dalam jangka waktu yang lama dengan aman.

DBMS adalah kumpulan data yang saling terkait dan seperangkat program untuk mengakses data tersebut. Pengumpulan data, biasanya disebut sebagai basis data, berisi informasi yang relevan dengan suatu perusahaan. Tujuan utama DBMS adalah menyediakan cara untuk menyimpan dan mengambil informasi basis data yang nyaman dan efisien.

Sistem basis data dirancang untuk mengelola banyak informasi. Manajemen data

melibatkan struktur pendefinisian untuk penyimpanan informasi dan menyediakan mekanisme untuk manipulasi informasi. Selain itu, sistem basis data harus memastikan keamanan informasi yang disimpan, meskipun sistem macet atau upaya akses yang tidak sah. Jika data akan dibagikan di antara beberapa pengguna, sistem harus menghindari kemungkinan hasil yang tidak normal.

DBMS diharapkan untuk:

1. Memungkinkan pengguna untuk membuat basis data baru dan menentukan skema basis data.
2. Memberi pengguna kemampuan untuk meminta data dan memodifikasi data, menggunakan bahasa *query*.
3. Mendukung penyimpanan data dalam jumlah yang sangat besar dan banyak dalam jangka waktu yang lama.
4. Memungkinkan akses yang efisien ke data untuk permintaan dan modifikasi basis data.
5. Mendukung pemulihan basis data dalam menghadapi kegagalan, banyak kesalahan, atau penyalahgunaan yang disengaja.
6. Kontrol akses ke data dari banyak pengguna sekaligus.

Dalam penerapannya, terdapat beberapa jenis software DBMS yang sering diaplikasikan untuk mengelola database perusahaan yaitu diantaranya:

1. MySQL

MySQL banyak digunakan di perusahaan karena memang tersedia secara gratis. Sehingga aplikasi ini cocok digunakan untuk bisnis-bisnis yang sedang berkembang. Meskipun tidak berbayar, namun tingkat keamanannya cukup baik dengan kecepatan akses data yang selalu stabil. Akan tetapi MySQL kurang kompatibel dengan bahasa pemograman Foxpro, Visual Basic (VB) dan Delphi serta kurang mampu menangani data yang jumlahnya terlalu besar.

2. Oracle
   Oracle merupakan perangkat lunak DBMS yang bagus dan berbayar. Oracle memiliki beragam fitur yang dapat memenuhi tuntutan fleksibilitas perusahaan besar. Bahkan oracle juga memiliki pemrosesan transaksi dengan performa yang sangat tinggi. Untuk memenuhi kriteria seperti pada pengertian DBMS, oracle tidak perlu diragukan lagi dalam hal keamanan.
3. Microsoft SQL Server
   Selain Oracle, Microsoft SQL Server juga cocok diaplikasikan pada sistem jaringan komputer perusahaan-perusahaan besar karena memiliki kemampuan mengelola data yang besar. Microsoft SQL Server memiliki sistem pengamanan data yang baik dan memiliki fitur back up, recovery dan rollback data. Namun, perangkat ini hanya bisa berjalan pada sistem operasi Windows saja.
4. Firebird
   Perangkat lunak DBMS lainnya adalah Firebird sebagai sistem manajemen basis

data yang relasional. Firebird menawarkan fitur yang sesuai dengan standar SQl-2003 dan ANSI SQL-99 serta dapat bekerja pada sistem operasi Windows dan Linux.

1.4. Aplikasi Sistem Basis Data

Telah banyak aplikasi yang menggunakan sistem basis data, terutama untuk sistem dengan banyak pertanyaan atau modifikasi yang dibuat. Berikut ini contoh beberapa aplikasi sistem basis data:

1. Informasi Perusahaan
    a. Penjualan: Untuk informasi pelanggan, produk, dan transaksi penjualan.
    b. Pembelian: Untuk informasi pemasok, dan transaksi pembelian
    c. Persediaan: Untuk informasi sisa stok, dan mutasi barang.
    d. Akuntansi: Untuk pembayaran, penerimaan, saldo akun, aset, dan informasi akuntansi lainnya.
    e. Sumber daya manusia: Untuk informasi tentang karyawan, gaji, pajak, dan tunjangan, dan untuk pembuatan gaji.
    f. Pabrikasi: Untuk manajemen rantai pasokan dan untuk melacak produksi barang di pabrik, inventaris barang di gudang dan toko, serta pesanan barang.
    g. Pengecer *online*: Untuk data penjualan ditambah pelacakan pesanan *online*, pembuatan daftar rekomendasi, dan pemeliharaan evaluasi produk *online*.
2. Perbankan dan Keuangan

a. Perbankan: Untuk informasi nasabah, rekening, pinjaman, dan transaksi perbankan.
b. Transaksi kartu kredit: Untuk pembelian kartu kredit dan pembuatan laporan bulanan.
c. Keuangan: Untuk menyimpan informasi tentang kepemilikan, penjualan, dan pembelian instrumen keuangan seperti saham dan obligasi; juga untuk menyimpan data pasar *real-time* untuk memungkinkan perdagangan *online* oleh pelanggan dan perdagangan otomatis oleh perusahaan.

3. Universitas: Untuk informasi mahasiswa, dosen, pendaftaran, mata kuliah, kelas, pengajaran dan nilai.
4. Maskapai Penerbangan: Untuk informasi pemesanan dan jadwal.
5. Telekomunikasi: Untuk menyimpan catatan panggilan yang dilakukan, menghasilkan tagihan bulanan, menjaga saldo pada kartu panggil prabayar, dan menyimpan informasi tentang jaringan komunikasi.

Dari daftar di atas dapat disimpulkan bahwa basis data merupakan bagian penting dari setiap perusahaan saat ini, tidak hanya menyimpan jenis informasi yang umum bagi sebagian besar perusahaan, tetapi juga informasi yang spesifik untuk kategori perusahaan[2].

[2] Henry F Korth dan Abraham Silberschatz, *Database System Concepts*, *Communications of the ACM*, vol. 40,

1.5. Tujuan Sistem Basis Data

Sistem basis data muncul sebagai tanggapan terhadap metode awal pengelolaan data komersial terkomputerisasi. Sebagai contoh sebuah universitas menyimpan informasi tentang semua dosen, mahasiswa, jurusan, dan mata kuliah. Salah satu cara untuk menyimpan informasi di komputer adalah menyimpannya dalam *file* sistem operasi. Untuk memungkinkan pengguna memanipulasi informasi, sistem memiliki sejumlah program aplikasi yang memanipulasi *file*, termasuk program untuk menambahkan mahasiswa baru, dosen, dan mata kuliah, menetapkan nilai, menghitung rata-rata IPK, dan menghasilkan transkrip nilai.

Pemrogram sistem menulis program aplikasi ini untuk memenuhi kebutuhan universitas. Program aplikasi baru ditambahkan ke sistem ketika diperlukan. Misalnya, sebuah universitas memutuskan untuk membuat jurusan baru (katakanlah, ilmu komputer). Akibatnya, universitas membuat jurusan baru dan membuat *file* permanen baru (atau menambahkan informasi ke *file* yang ada) untuk merekam informasi tentang semua dosen, mahasiswa, dan mata kuliah pada jurusan tersebut. Universitas harus menulis program aplikasi baru untuk berurusan dengan aturan khusus untuk jurusan baru. Program aplikasi baru mungkin juga harus ditulis untuk menangani aturan baru di universitas. Dengan demikian, seiring berjalannya waktu,

---

1997,
http://portal.acm.org/citation.cfm?doid=253671.253760.

sistem memperoleh lebih banyak *file* dan lebih banyak program aplikasi.

Sistem pemrosesan *file* yang khas ini didukung oleh sistem operasi konvensional. Sistem menyimpan catatan permanen dalam berbagai *file*, dan membutuhkan program aplikasi yang berbeda untuk mengekstrak catatan dari, dan menambahkan catatan ke, *file* yang sesuai. Sebelum sistem manajemen basis data (DBMS) diperkenalkan, organisasi biasanya menyimpan informasi dalam sistem tersebut.

Menyimpan informasi organisasi dalam sistem pemrosesan *file* memiliki sejumlah kelemahan utama:

1. Redundansi dan inkonsistensi data.
   Karena *programmer* yang berbeda membuat *file* dan program aplikasi dalam jangka waktu yang lama, berbagai *file* cenderung memiliki struktur yang berbeda dan program dapat ditulis dalam beberapa bahasa pemrograman. Selain itu, informasi yang sama dapat digandakan di beberapa tempat (*file*). Misalnya, jika seorang mahasiswa mengambil jurusan lebih dari satu (misalnya ilmu komputer dan akuntansi) alamat dan nomor telepon mahasiswa tersebut dapat muncul dalam *file* yang terdiri dari catatan mahasiswa di jurusan ilmu komputer dan dalam *file* yang terdiri dari catatan mahasiswa di jurusan akuntansi. Redundansi ini menyebabkan penyimpanan lebih tinggi dan biaya akses. Selain itu, ini dapat menyebabkan inkonsistensi data, misalnya,

alamat mahasiswa yang diubah dalam catatan jurusan ilmu komputer tetapi tidak berubah di tempat lain dalam sistem.

2. Kesulitan dalam mengakses data.
   Misalkan salah satu ketua jurusan di universitas perlu mencari tahu nama-nama semua mahasiswa yang tinggal dalam area kode pos tertentu. Karena para perancang sistem asli tidak mengantisipasi permintaan ini, tidak ada program aplikasi yang dapat memenuhi permintaan tersebut. Namun, ada program aplikasi untuk menghasilkan daftar semua mahasiswa. Petugas universitas sekarang memiliki dua pilihan: mendapatkan daftar semua mahasiswa dan mengekstrak informasi yang diperlukan secara manual atau meminta programmer untuk menulis program aplikasi yang diperlukan. Kedua alternatif itu jelas tidak memuaskan. Misalkan program semacam itu ditulis, dan bahwa, beberapa hari kemudian, pegawai yang sama memerlukan daftar untuk mahasiswa yang telah mengambil minimal 60 sks, maka sekali lagi, petugas memiliki dua opsi sebelumnya, yang keduanya tidak memuaskan.
   Intinya di sini adalah bahwa lingkungan pemrosesan *file* konvensional tidak memungkinkan data yang diperlukan untuk diambil dengan cara yang nyaman dan efisien. Diperlukan sistem pengambilan data yang lebih responsif untuk penggunaan umum.
3. Isolasi data

Karena data tersebar di berbagai *file*, dan *file* mungkin dalam format yang berbeda, sulit untuk menulis program aplikasi baru untuk mengambil data yang sesuai.

4. Masalah integritas
   Nilai data yang disimpan dalam database harus memenuhi beberapa jenis batasan konsistensi. Misalkan universitas memiliki akun untuk setiap jurusan, dan mencatat jumlah saldo di setiap akun. Misalkan juga bahwa universitas mensyaratkan bahwa saldo akun suatu jurusan tidak boleh di bawah nol. Pengembang menetapkan kendala ini dalam sistem dengan menambahkan kode yang sesuai di berbagai program aplikasi. Namun, ketika kendala baru ditambahkan, sulit untuk mengubah programnya lagi. Masalahnya diperparah ketika kendala melibatkan beberapa item data dari *file* yang berbeda.
5. Masalah atomisitas
   Sistem komputer, seperti perangkat lainnya, dapat mengalami kegagalan. Dalam banyak aplikasi, sangat penting bahwa, jika terjadi kegagalan, data disimpan kembali ke keadaan konsisten yang ada sebelum kegagalan. Pertimbangkan program untuk mentransfer 50 juta dari saldo akun departemen A ke saldo akun departemen B. Jika terjadi kegagalan sistem selama pelaksanaan program, ada kemungkinan 50 juta dihapus dari saldo departemen A tetapi tidak ditambahkan ke saldo departemen B, menghasilkan kondisi basis data yang tidak

konsisten. Jelas, sangat penting untuk konsistensi basis data bahwa kredit dan debit terjadi, atau tidak terjadi.

Artinya, transfer dana harus bersifat atomik — itu harus terjadi secara keseluruhan atau tidak sama sekali. Sulit untuk memastikan atomisitas dalam sistem pemrosesan *file* konvensional.

6. Anomali akses bersamaan

   Demi keseluruhan kinerja sistem dan respons yang lebih cepat, banyak sistem memungkinkan banyak pengguna untuk memperbarui data secara bersamaan. Saat ini, pengecer internet terbesar mungkin memiliki jutaan akses per hari ke data mereka oleh pembeli. Dalam lingkungan seperti itu, interaksi pembaruan bersamaan dimungkinkan dan dapat menghasilkan data yang tidak konsisten.

   Sebagai contoh, anggaplah sebuah program pendaftaran memelihara jumlah mahasiswa yang terdaftar untuk suatu mata kuliah. Ketika seorang siswa mendaftar, program membaca hitungan saat ini untuk kelas, memverifikasi bahwa jumlah belum mencapai batas, menambahkan satu ke daftar, dan menyimpan hitungan kembali dalam basis data. Misalkan dua siswa mendaftar secara bersamaan, dengan hitungan di 39. Kedua eksekusi program dapat membaca nilai 39, dan keduanya kemudian akan menulis kembali 40, yang mengarah ke peningkatan yang salah hanya 1, meskipun dua siswa berhasil mendaftar

untuk kursus dan hitungannya harus 41. Selanjutnya, anggaplah batas registrasi kursus adalah 40, dalam kasus di atas kedua siswa akan dapat mendaftar, yang mengarah pada pelanggaran batas 40 siswa.

7. Masalah keamanan.
   Tidak semua pengguna sistem basis data harus dapat mengakses semua data. Misalnya, di universitas, personel penggajian hanya perlu melihat bagian database yang memiliki informasi keuangan. Mereka tidak memerlukan akses ke informasi tentang catatan akademik. Tetapi, karena program aplikasi ditambahkan ke sistem pemrosesan *file*, menegakkan batasan keamanan semacam itu sangatlah sulit.

Adapun kelebihan dari menggunakan basis data adalah:

1. Data dapat dibagikan
   Membagikan data berarti tidak hanya aplikasi dapat digunakan bersama, tetapi juga aplikasi yang baru dapat dibangun dengan menggunakan data yang sama. Artinya, programmer dapat membuat aplikasi baru tanpa harus membuat basis data yang baru lagi.
2. Redundansi dapat dikurangi
   Dengan adanya batasan kunci yang diatur dalam basis data, memungkinkan data untuk mengecek yang sudah tersimpan dan tidak menyimpan data yang sama.
3. Ketidak konsistenan dapat dihindari

Karena data disimpan hanya satu kali dalam basis data, sehingga perubahan pada data tidak perlu di beberapa tempat.

4. Dukungan transaksi dapat diberikan
   Basis data dengan relasi antar tabel akan secara otomatis mengubah data pada tabel yang berhubungan, sehingga transaksi yang terjadi tidak terputus.
5. Integritas bisa dijaga
   Masalah integritas adalah memastikan data yang disimpan di dalam basis data adalah data yang benar. Dengan adanya penanganan terhadap redundansi, maka integritas data di dalam basis data dapat terjaga.
6. Keamanan dapat ditegakkan
   Dalam sistem basis data dapat dibatasi hak akses pengguna, sehingga pengguna hanya dapat mengakses data yang bersangkutan dengan pekerjaannya dan tidak bisa mengakses data lain yang dibatasi.
7. Persyaratan yang saling bertentangan dapat seimbang
   Dalam organisasi, akan muncul persyaratan-persyaratan penggunaan data yang saling bertentangan, untuk itu sistem basis data dapat menetapkan akses cepat untuk sistem yang penting atau yang diutamakan.
8. Standar dapat ditegakkan
   Dengan kontrol terpusat, *administrator* basis data dapat menetapkan standar terhadap data

yang direpresentasikan dari basis data, misalnya untuk nama tabel dan header[3].

1.6. Bahasa Basis Data

Basis data menyediakan *Data-Definition Language* (DDL) untuk menentukan skema basis data dan *Data-Manipulation Language* (DML) untuk mengekspresikan permintaan dan pembaruan basis data.

1. *Data-Definition Language* (DDL)
   DDL digunakan untuk menentukan properti tambahan dari data. DDL disebut penyimpanan data dan bahasa definisi. DDL adalah perintah-perintah yang biasa digunakan oleh *Database Administrator* (DBA). Pernyataan-pernyataan ini menentukan detail implementasi dari skema basis data, yang biasanya disembunyikan dari pengguna. Nilai data yang disimpan dalam basis data harus memenuhi batasan konsistensi tertentu. Misalnya, misalkan universitas mengharuskan saldo akun suatu departemen tidak boleh negatif. DDL menyediakan fasilitas untuk menentukan batasan seperti itu. Sistem basis data memeriksa kendala ini setiap kali basis data diperbarui.
   Dengan bahasa ini dapat dibuat tabel baru, membuat indeks, mengubah tabel,

[3] Hector Garcia-Molina, Jeffrey D. Ullman, dan Jennifer Widom, *DATABASE SYSTEMS The Complete Book*, *Pearson Prentice Hall*, vol. 26 (New Jersey: PEARSON, 2009), http://www.worldcat.org/isbn/813170842X.

menentukan struktur tabel, dll. Hasil dari kompilasi perintah DDL menjadi Kamus Data, yaitu data yang menjelaskan data sesungguhnya. Perintah dasar yang termasuk ke dalam DDL yaitu :

- *Create*, digunakan untuk membuat *database*, tabel, *view* dan kolom.
- *Alter*, digunakan untuk mengubah struktur tabel yang telah dibuat seperti mengganti nama tabel, menambah, mengubah dan menghapus kolom pada tabel.
- *Rename*, digunakan untuk mengubah nama objek.
- *Drop*, digunakan untuk menghapus *database* dan menghapus tabel.

Contoh:

```
create table jurusan
(nama_jurusan char (20),
lokasi char (15),
anggaran numeric (12,2));
```

2. *Data-Manipulation Language* (DML)
   DML adalah bahasa yang memungkinkan pengguna untuk mengakses atau memanipulasi data sebagaimana diatur oleh model data yang sesuai. DML juga digunakan untuk memasukkan, merubah, dan menghapus data-data di dalam sebbuah tabel. Perintah yang termasuk ke dalam DML yaitu:

- *Insert*, digunakan untuk menyisipkan atau memasukkan data baru ke dalam tabel.
- *Update*, digunakan untuk memperbaharui data lama menjadi data baru.
- *Delete*, digunakan untuk menghapus data dari tabel.
- *Select*, digunakan untuk menampilkan data dari satu tabel atau beberapa tabel dalam relasi.

Contoh:

```
select dosen,nama
from dosen
where dosen.jurusan = 'Fisika';
```

3. Data Control Language (DCL)

DCL adalah bahasa SQL yang berkaitan dengan manipulasi dan hak akses pengguna. Perintah yang termasuk ke dalam DCL yaitu:

- *Grant*, digunakan untuk memberikan hak akses oleh *administrator* kepada pengguna biasa.
- *Revoke*, digunakan untuk menghilangkan atau mencabut hak akses yang telah diberikan oleh *administrator* kepada pengguna biasa.

1.7. Latihan

1. Apa yang dimaksud dengan Sistem Manajemen Basis Data?

2. Software apa saja yang termasuk Sistem Manajemen Basis Data? Jelaskan!
3. Apa saja kelemahan dari penyimpanan file pada sistem operasi?
4. Apa keuntungan yang bisa didapatkan dari menggunakan Sistem Manajemen Basis Data?
5. Apa saja jenis bahasa yang digunakan dalam Basis Data?

BAB II

Basis Data Relasional

2.1. Model Data

Model data adalah notasi untuk menggambarkan data atau informasi. Model data umumnya terdiri dari tiga bagian:

1. Struktur data.
   Alat-alat dalam bahasa pemrograman seperti C atau Java biasa digunakan untuk menggambarkan struktur data yang digunakan oleh suatu program: *array* dan struktur ("*struct*") atau objek, misalnya. Struktur data yang digunakan untuk mengimplementasikan data di komputer, dalam diskusi sistem basis data disebut sebagai model data fisik, meskipun sebenarnya jauh dari benar-benar berfungsi sebagai implementasi fisik data. Dalam dunia basis data, model data berada pada tingkat yang agak lebih tinggi dari struktur data, dan kadang-kadang disebut sebagai model konseptual untuk menekankan perbedaan tingkat.
2. Operasi pada data.
   Dalam bahasa pemrograman, operasi pada data umumnya adalah segala sesuatu yang dapat diprogram. Dalam model data basis data, biasanya ada serangkaian operasi terbatas yang dapat dilakukan. Pengguna umumnya diizinkan untuk melakukan serangkaian pertanyaan terbatas (operasi

yang mengambil informasi) dan modifikasi (operasi yang mengubah database). Keterbatasan ini bukan kelemahan, tetapi kekuatan. Dengan membatasi operasi, dimungkinkan bagi programmer untuk menggambarkan operasi basis data pada tingkat yang sangat tinggi, namun memiliki sistem manajemen basis data yang melaksanakan operasi secara efisien. Sebagai perbandingan, umumnya tidak mungkin untuk mengoptimalkan program dalam bahasa konvensional seperti C, sepanjang algoritma yang tidak efisien (misalnya, bubblesort) diganti dengan yang lebih efisien. (mis., quicksort).

3. Kendala pada data.
   Model data basis data biasanya memiliki cara untuk menggambarkan batasan pada data. Batasan ini dapat berkisar dari yang sederhana (mis., "Satu hari dalam seminggu adalah bilangan bulat antara 1 dan 7" atau "sebuah film memiliki paling banyak satu judul") hingga beberapa batasan yang sangat kompleks[4].

2.2. Struktur Basis Data Relasional

Basis data relasional terdiri dari kumpulan tabel, yang masing-masing diberi nama unik. Sebagai contoh, perhatikan tabel dosen pada tabel 2.1, yang menyimpan informasi tentang dosen. Tabel ini memiliki empat kolom: ID, nama,

[4] Satinder Bal Gupta dan Aditya Mittal, *Database Management System* (UNIVERSITY SCIENCE PRESS, 2017).

jurusan, dan gaji. Setiap baris tabel ini mencatat informasi tentang dosen, yang terdiri dari ID dosen, nama, jurusan, dan gaji. Begitu pula dengan mata kuliah, tabel 2.2 menyimpan informasi tentang mata kuliah, yang terdiri dari kode mata kuliah, judul mata kuliah, jurusan, dan sks, untuk setiap mata kuliah. Perhatikan bahwa setiap dosen diidentifikasi oleh nilai ID, sedangkan setiap mata kuliah diidentifikasi oleh nilai kode.

Tabel 2.1. Tabel Dosen

ID	Nama	Jurusan	Gaji
342355	Suparni	Ilmu Komputer	5.500.000
213233	Andri	Sistem Informasi	3.500.000
535353	Reza	Biologi	4.800.000
212124	Bobby	Sistem Informasi	5.500.000
321233	Sulastri	Matematika	4.500.000

Tabel 2.2. Tabel Mata Kuliah

Kode_MK	Judul	Nama_Jurusan	SKS
IK-101	Basis Data	Ilmu Komputer	2
IK-115	Penambangan Data	Ilmu Komputer	2
IK-173	Pemrograman Visual	Ilmu Komputer	3
SI-101	Tata Kelola TI	Sistem Informasi	3

SI-202	Audit TI	Sistem Informasi	2

Tabel 2.3. Tabel Prasyarat

Kode_MK	Kode_Prasyarat
IK-115	IK-101
IK-173	IK-101
SI-202	SI-101

Tabel 2.3 menunjukkan tabel prasyarat, yang menyimpan mata kuliah prasyarat untuk setiap mata kuliah. Tabel ini memiliki dua kolom, kode mata kuliah dan kode prasyarat. Setiap baris terdiri dari sepasang pengidentifikasi saja sehingga mata kuliah kedua merupakan prasyarat untuk mata kuliah pertama.

Dengan demikian, satu baris dalam tabel prasyarat menunjukkan bahwa dua mata kuliah terkait dalam arti bahwa satu mata kuliah merupakan prasyarat bagi mata kuliah lainnya.

2.3. Skema Basis Data

Ketika berbicara tentang suatu basis data, harus dibedakan antara skema basis data, yang merupakan desain logis dari basis data, dan contoh basis data, yang merupakan *snapshot* dari data dalam basis data pada waktu tertentu.

Konsep relasi atau tabel sesuai dengan gagasan bahasa pemrograman suatu variabel, sedangkan konsep skema tabel sesuai dengan gagasan definisi tipe bahasa pemrograman. Nilai

variabel yang diberikan dapat berubah seiring waktu, demikian pula isi dari tabel dapat berubah seiring waktu ketika tabel diperbarui. Sebaliknya, skema hubungan umumnya tidak berubah.

Tabel 2.4. Tabel Jurusan

Nama_Jurusan	Lokasi	Anggaran
Sistem Informasi	Kampus I	200.000.000
Ilmu Komputer	Kampus I	250.000.000
Biologi	Kampus I	150.000.000
Pendidikan Biologi	Kampus II	200.000.000
Matematika	Kampus I	175.000.000
Pendidikan Matematika	Kampus II	125.000.000

Pertimbangkan hubungan jurusan pada Tabel 2.4. Skema untuk tabel itu adalah jurusan (jurusan, lokasi, anggaran). Perhatikan bahwa atribut nama jurusan muncul di skema dosen dan skema jurusan. Duplikasi ini bukan kebetulan. Sebaliknya, menggunakan atribut umum dalam skema hubungan adalah salah satu cara untuk menghubungkan hubungan yang berbeda.

Sebagai contoh, misalkan pengguna ingin mencari informasi tentang semua dosen yang bekerja di kampus I. Pengguna melihat pertama pada tabel jurusan untuk menemukan semua jurusan yang bertempat di Kampus I. Kemudian, untuk setiap jurusan seperti itu, pengguna mencari di dalam tabel dosen untuk menemukan informasi tentang dosen yang terkait dengan jurusan yang sesuai.

2.4. Kunci

Harus ada cara untuk menentukan bagaimana data dalam tabel tertentu dibedakan. Ini dinyatakan dalam atribut mereka. Artinya, nilai-nilai atribut nilai data harus sedemikian rupa sehingga mereka dapat mengidentifikasi baris data secara unik. Dengan kata lain, tidak ada dua *record* data dalam suatu tabel yang diizinkan memiliki nilai yang persis sama untuk semua atribut.

1. *Superkey*
   adalah sekumpulan satu atau lebih atribut yang diambil secara kolektif, memungkinkan untuk mengidentifikasi *record* data unik dalam tabel. Sebagai contoh, atribut ID dari tabel dosen cukup untuk membedakan satu dosen dari yang lain. Jadi, ID adalah *superkey*. Atribut nama dosen, di sisi lain, bukan *superkey*, karena beberapa dosen mungkin memiliki nama yang sama. *Superkey* mungkin berisi atribut yang tidak berhubungan. Misalnya, kombinasi ID dan nama adalah *superkey* untuk tabel dosen. Jika ID adalah *superkey*, maka *superkeys* semacam itu disebut kunci kandidat (*candidate key*).
2. Kunci kandidat (*Candidate Key*)
   Ada kemungkinan bahwa beberapa set atribut yang berbeda dapat berfungsi sebagai kunci kandidat. Misalkan kombinasi nama dan jurusan cukup untuk membedakan antara anggota tabel dosen. Kemudian, ID dan jurusan adalah kunci kandidat. Meskipun

atribut ID dan nama bersama-sama dapat membedakan *record* dosen, kombinasinya, {ID, Nama}, tidak membentuk kunci kandidat, karena atribut ID sendiri adalah kunci kandidat.

3. Kunci Utama (*Primary* Key)
   Istilah kunci utama (*primary key*) digunakan untuk menunjukkan kunci kandidat yang dipilih oleh perancang basis data sebagai sarana utama untuk mengidentifikasi data dalam suatu tabel. Kunci (baik utama, kandidat, atau super) adalah properti dari seluruh tabel, dan bukan data individual. Setiap dua data individu dalam tabel dilarang memiliki nilai yang sama pada atribut kunci pada saat yang sama. Penunjukan kunci mewakili kendala dalam perusahaan dunia nyata yang dimodelkan. Kunci primer harus dipilih dengan hati-hati. Seperti yang tercatat, nama seseorang jelas tidak cukup, karena mungkin ada banyak orang dengan nama yang sama. Kunci utama harus dipilih sedemikian rupa sehingga nilai atributnya tidak pernah, atau sangat jarang, berubah. Misalnya, bidang alamat seseorang tidak boleh menjadi bagian dari kunci utama, karena kemungkinan akan berubah. Nomor KTP, misalnya, dijamin tidak akan pernah berubah. Pengidentifikasi unik yang dihasilkan oleh perusahaan umumnya tidak berubah, kecuali jika dua perusahaan bergabung, dalam kasus seperti itu pengidentifikasi yang sama mungkin telah dikeluarkan oleh kedua perusahaan, dan

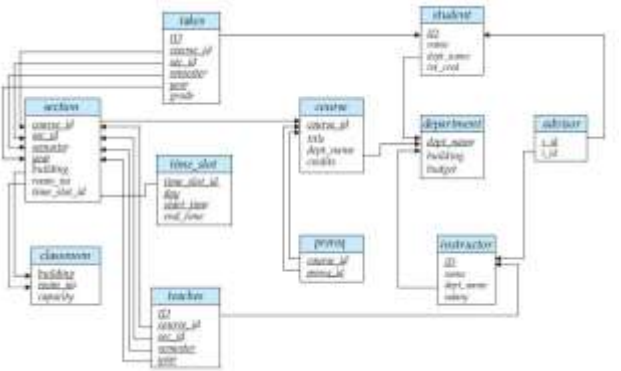
realokasi pengidentifikasi mungkin diperlukan untuk memastikan mereka unik. Merupakan kebiasaan untuk membuat daftar atribut kunci utama dari skema tabel sebelum atribut lainnya; misalnya, atribut jurusan terdaftar terlebih dahulu, karena itu adalah kunci utama. Atribut kunci primer juga digarisbawahi.

4. Kunci asing (*foreign key*)
   Suatu tabel, katakanlah pada tabel kelas, dapat memasukkan di antara atribut-atributnya kunci utama dari tabel lain, misalnya, atribut jurusan dalam dosen adalah kunci asing (*foreign key*) dari dosen, referensi jurusan, karena nama jurusan adalah kunci utama dari tabel jurusan.

2.5. Skema Diagram

Skema basis data, bersama dengan dependensi *primary key* dan *foreign key*, dapat digambarkan oleh diagram skema. Gambar 2.1 menunjukkan diagram skema untuk organisasi universitas. Setiap tabel muncul sebagai kotak, dengan nama tabel di bagian atas, dan atribut yang tercantum di dalam kotak. Atribut kunci primer adalah yang bergaris bawah. Ketergantungan kunci asing muncul sebagai

panah dari atribut kunci asing dari tabel referensi ke kunci utama dari tabel yang direferensikan.



Gambar 2.1. Skema Diagram untuk Basis Data Universitas

2.6. Latihan

1. Jelaskan bagian-bagian dari model data.
2. Jelaskan perbedaan antara skema basis data dan contoh basis data.
3. Jelaskan perbedaan antara *superkey, candidate key, primary key* dan *foreign key*.

BAB III

Structured Query Language (SQL)

3.1. SQL Data Definition Language (DDL)

Himpunan tabel dalam basis data harus ditentukan ke sistem dengan menggunakan *Data Definition Language* (DDL). SQL DDL memungkinkan spesifikasi tidak hanya seperangkat hubungan, tetapi juga informasi tentang masing-masing hubungan, termasuk:

- Skema untuk setiap hubungan.
- Jenis nilai yang terkait dengan setiap atribut.
- Kendala integritas.
- Kumpulan indeks yang harus dipertahankan untuk setiap tabel.
- Informasi keamanan dan otorisasi untuk setiap tabel.
- Struktur penyimpanan fisik dari setiap tabel pada disk.

3.1.1. Tipe Dasar Variabel SQL

Standar SQL mendukung berbagai tipe bawaan, termasuk:

- ***char* (n):** *String* karakter dengan panjang tetap dengan panjang yang ditentukan pengguna (n). Misalnya Nomor KTP, karena jumlah karakternya pasti sama sesuai menggunakan char.
- ***varchar* (n):** *String* karakter panjang variabel dengan panjang maksimum yang ditentukan pengguna (n). Misalnya nama, jumlah

karakter bisa saja berbeda sehingga cukup menentukan jumlah maksimal Panjang karakter.

- ***int*: *Integer*** atau bilangan bulat.
- ***smallint*:** *Integer* kecil (subset yang bergantung pada mesin dari tipe integer)
- ***numerik* (p, d):** Nomor titik tetap dengan presisi yang ditentukan pengguna. Angka terdiri dari p digit (ditambah tanda), dan d digit p berada di kanan titik desimal. Dengan demikian, numerik (3,1) memungkinkan 44,5 disimpan dengan tepat, tetapi 444,5 atau 0,32 tidak dapat disimpan dengan tepat di bidang jenis ini.
- ***real, double*:** angka floating-point dan presisi ganda, bisa menampung hingga 15 digit pecahan.
- **float (n):** Angka titik-mengambang, dengan ketepatan setidaknya n digit.

Setiap jenis dapat menyertakan nilai khusus yang disebut nilai *null*. Nilai *null* menunjukkan nilai kosong yang mungkin ada tetapi tidak diketahui, atau yang mungkin tidak ada sama sekali. Dalam kasus tertentu, mungkin saja nilai *null* dilarang untuk dimasukkan, maka dapat dibuat variabel yang *not null*.

Tipe data *char* menyimpan *string* panjang tetap. Pertimbangkan, misalnya, atribut A dari tipe *char* (10). Jika *string* "Ada" disimpan dalam atribut ini, 7 spasi ditambahkan ke *string* untuk membuatnya menjadi 10 karakter. Sebaliknya, jika atribut B bertipe *varchar* (10), dan "Ada" disimpan di atribut B, tidak ada spasi yang akan ditambahkan. Ketika membandingkan dua nilai

tipe *char*, jika mereka memiliki panjang yang berbeda, ruang tambahan secara otomatis ditambahkan ke yang lebih pendek untuk membuat ukuran yang sama.

3.1.2. Definisi Skema Dasar

Hubungan SQL didefenisikan dengan menggunakan perintah *create table*. Perintah berikut ini membuat tabel jurusan dalam basis data.

```
create table jurusan
(nama_jurusan varchar (20),
lokasi varchar (15),
anggaran numeric (12,2),
primary key (nama_jurusan));
```

Tabel yang dibuat di atas memiliki tiga atribut, nama jurusan, yang merupakan karakter *string* dengan panjang maksimum 20, lokasi, yang merupakan karakter *string* dengan panjang maksimum 15, dan anggaran, yang merupakan angka dengan total 12 digit, 2 di antaranya adalah setelah titik desimal. Perintah *create table* juga menentukan bahwa atribut nama jurusan adalah kunci utama tabel jurusan. Tanda titik koma yang ditunjukkan pada akhir membuat pernyataan tabel, serta akhir pernyataan SQL.

```
create table jurusan
(nama_jurusan varchar (20),
lokasi varchar (15),
anggaran numeric (12,2),
```

```
primary key (nama_jurusan));


create table mata_kuliah
(kode_MK varchar (7),
judul varchar (50),
nama_jurusan varchar (20),
sks numeric (2,0),
primary key (kode_MK),
foreign  key  (nama_jurusan)
references jurusan);


create table dosen
(ID varchar (5),
nama varchar (20) not null,
nama_jurusan varchar (20),
gaji numeric (8,2),
primary key (ID),
foreign  key  (nama_jurusan)
references jurusan);


create table kelas
(kode_MK varchar (8),
kode_kelas varchar (8),
semester varchar (6),
tahun numeric (4,0),
lokasi varchar (15),
ruangan varchar (7),
waktu varchar (4),
primary    key    (kode_MK,
kode_kelas, semester, tahun),
foreign    key    (kode_MK)
references mata_kuliah);
```

```
create table pengajaran
(ID varchar (5),
kode_MK varchar (8),
kode_kelas varchar (8),
semester varchar (6),
tahun numeric (4,0),
primary key (ID, kode_MK,
kode_kelas, semester, tahun),
foreign key (kode_MK,
kode_kelas, semester, tahun)
references kelas,
foreign key (ID) references
dosen);
```

SQL mendukung sejumlah kendala integritas yang berbeda, di antaranya:

1. ***Primary key* (Aj_1, Aj_2,.,, Aj_m)**: Spesifikasi primary-key mengatakan bahwa atribut Aj_1, Aj_2,. . ., Aj_m membentuk kunci utama untuk tabel. Atribut *primary key* harus *not null* dan unik; yaitu, tidak ada *record* yang dapat memiliki nilai nol untuk atribut *primary key*, dan tidak ada dua *record* dalam tabel yang dapat sama pada semua atribut *primary key*. Meskipun spesifikasi kunci primer bersifat opsional, merupakan ide bagus untuk menentukan kunci primer untuk setiap tabel.
2. ***Foreign key* (Ak_1, Ak_2, ..., Ak_n) *references* s**: Spesifikasi kunci asing mengatakan bahwa nilai atribut (Ak_1, Ak_2, ..., Ak_n) untuk setiap *record* dalam tabel harus sesuai dengan nilai-nilai dari atribut kunci utama dari beberapa *record* dalam hubungan.

3. ***Not null***: Batasan *not null* pada atribut menetapkan bahwa nilai nol tidak diizinkan untuk atribut itu; dengan kata lain, batasan mengecualikan nilai nol dari domain atribut itu. Sebagai contoh, batasan *not null* pada atribut nama dari tabel dosen, ini memastikan bahwa nama dosen tidak boleh kosong.

SQL mencegah pembaruan apa pun ke basis data yang melanggar batasan integritas. Sebagai contoh, jika *record* yang baru saja dimasukkan atau dimodifikasi dalam suatu tabel memiliki nilai nol untuk atribut *primary key*, atau jika *record* memiliki nilai yang sama pada atribut *primary key* seperti halnya *record* lain dalam tabel, SQL menandai kesalahan dan mencegah pembaruan. Demikian pula, penyisipan data mata kuliah dengan nilai nama jurusan yang tidak muncul dalam tabel jurusan akan melanggar batasan pada kunci asing, dan SQL akan mencegah penyisipan tersebut terjadi.

Tabel yang baru dibuat awalnya kosong. Perintah *insert* bisa digunakan untuk menambahkan data ke dalam tabel. Misalnya, jika ingin memasukkan fakta bahwa ada seorang dosen bernama Ulfayani di jurusan Biologi dengan ID Dosen 10211 dan gaji 3.000.000, maka perintah yang dibuat:

```
insert into dosen
values (10211, 'Ulfayani',
'Biologi', 3000000);
```

Nilai ditentukan dalam urutan di mana atribut yang sesuai tercantum dalam skema tabel. Perintah *delete* dapat digunakan untuk menghapus data dari suatu tabel, dengan perintah sebagai berikut:

```
delete from mahasiswa;
```

Perintah di atas akan menghapus semua *record* dari tabel mahasiswa. Bentuk lain dari perintah *delete* memungkinkan *record* tertentu untuk dihapus yang akan dibahas pada bab berikutnya. Untuk menghapus tabel dari *database* SQL, perintah *drop table* dapat digunakan.

```
drop table mahasiswa;
```

Perintah *drop table* menghapus semua informasi tentang tabel yang dihapus dari *database*. Perintah itu adalah tindakan yang lebih drastis daripada

```
delete from mahasiswa;
```

Perintah di atas mempertahankan tabel mahasiswa, tetapi menghapus semua *record* di dalam tabel mahasiswa. Perintah *drop* menghapus tidak hanya semua *record*, tetapi juga skema untuk tabel mahasiswa. Setelah tabel mahasiswa dihapus, tidak ada *record* data yang dapat dimasukkan ke dalam tabel mahasiswa kecuali dibuat kembali dengan perintah *create table*.

Perintah *alter table* dapat digunakan untuk menambahkan atribut ke tabel yang ada. Semua *record* dalam tabel tersebut ditetapkan *null* sebagai nilai untuk atribut baru. Bentuk perintah *alter table* adalah:

```
alter table dosen add alamat varchar(50);
```

dimana dosen adalah nama dari tabel yang ada, alamat adalah nama atribut yang akan ditambahkan, dan varchar(50) adalah tipe atribut yang ditambahkan. Atribut dari suatu tabel dapat dihilangkan dengan perintah:

```
alter table dosen drop alamat;
```

dimana dosen adalah nama tabel yang ada, dan alamat adalah nama atribut tabel yang akan dihapus.

3.2. Kueri pada Satu Tabel

Struktur dasar kueri SQL terdiri dari tiga klausa: *select, from,* and *where*. Permintaan mengambil sebagai input tabel yang tercantum dalam klausa *from*, mengoperasikannya seperti yang ditentukan pada klausa *select* dan *where*, dan kemudian menghasilkan tabel sebagai hasilnya.

Perhatikan pertanyaan sederhana menggunakan contoh basis data universitas, "Temukan nama semua dosen". Nama dosen muncul di atribut nama, jadi dimasukkan ke dalam klausa *select*.

```
select nama from dosen;
```

Tabel 3.1. Hasil Pencarian Nama Dosen

Nama
Suparni
Andri
Reza
Bobby
Sulastri

Sekarang pertimbangkan permintaan lain, "Temukan nama jurusan semua dosen," yang dapat ditulis sebagai:

```
select nama_jurusan from dosen;
```

Tabel 3.2. Hasil Pencarian Jurusan Dosen

Nama_jurusan
Ilmu Komputer
Sistem Informasi
Biologi
Sistem Informasi
Matematika

Karena lebih dari satu dosen dapat menjadi bagian dari sebuah jurusan, nama jurusan dapat muncul lebih dari satu kali dalam tabel dosen. Hasil kueri di atas adalah tabel yang berisi nama-nama jurusan.

Dalam definisi matematika formal dari model relasional, suatu tabel adalah himpunan. Dengan demikian, duplikat *record* tidak akan

pernah muncul dalam tabel. Dalam praktiknya, penghapusan duplikat memakan waktu. Oleh karena itu, SQL memungkinkan duplikat dalam tabel maupun dalam hasil ekspresi SQL. Dengan demikian, kueri SQL sebelumnya mencantumkan setiap nama jurusan satu kali untuk setiap *record* yang muncul dalam tabel dosen.

Dalam kasus di mana pengguna ingin memaksakan penghapusan duplikat, kata kunci berbeda setelah *select* dapat disisipkan. Kueri dapat ditulis sebagai berikut:

```
    select distinct nama_jurusan
from dosen;
```

Tabel 3.3. Hasil Pencarian Jurusan Dosen dengan *distinct*

Nama_Jurusan
Ilmu Komputer
Sistem Informasi
Biologi
Matematika

Jika duplikat ingin dihapus. Hasil kueri di atas akan berisi setiap nama jurusan paling banyak satu kali. SQL menggunakan kata kunci *all* digunakan untuk menentukan secara eksplisit bahwa duplikat tidak dihapus:

```
    select all nama_jurusan from
dosen;
```

Klausa *select* juga dapat berisi ekspresi aritmatika yang melibatkan operator +, -, ∗, dan / yang beroperasi pada konstanta atau atribut *record*. Misalnya:

```
select ID, nama,
nama_jurusan, gaji * 1.1
from dosen;
```

Ini menunjukkan apa yang akan terjadi jika memberikan kenaikan 10% untuk setiap dosen, bagaimanapun, kueri ini tidak menghasilkan perubahan apa pun pada tabel dosen.

SQL juga menyediakan tipe data khusus, seperti berbagai bentuk tipe tanggal, dan memungkinkan beberapa fungsi aritmatika beroperasi pada tipe ini. Klausa *where* memungkinkan untuk hanya memilih baris-baris dalam tabel hasil dari klausa *from* yang memenuhi predikat tertentu. Pertimbangkan kueri "Temukan nama semua dosen di jurusan Sistem Informasi yang memiliki gaji lebih dari 4.000.000" Kueri ini dapat ditulis dalam SQL sebagai:

```
select nama from dosen
where nama_jurusan = 'Sistem
Informasi' and gaji >
4000000;
```

SQL memungkinkan penggunaan penghubung logis *and*, *or*, dan *not* pada klausa *where*. Operan dari penghubung logis dapat berupa ekspresi yang melibatkan operator

pembanding <, <=, >, > =, =, dan <>. SQL memungkinkan menggunakan operator pembanding untuk membandingkan *string* dan ekspresi aritmatika, serta tipe khusus, seperti tipe tanggal.

3.3. Kueri pada Multi Tabel

Sejauh ini contoh kueri yang sudah dipelajari berada pada satu tabel. Permintaan sering kali perlu mengakses informasi dari berbagai hubungan. Sebagai contoh, misalkan pengguna ingin menjawab permintaan "Ambil nama semua dosen, bersama dengan nama jurusan dan lokasi jurusannya."

Melihat skema tabel dosen, pengguna menyadari bahwa bisa saja mendapatkan nama jurusan dari atribut jurusan, tetapi lokasi jurusan ada di atribut lokasi dari tabel jurusan. Untuk menjawab pertanyaan, setiap *record* dalam tabel dosen harus dicocokkan dengan *record* dalam tabel jurusan yang nilai nama jurusannya cocok dengan nilai nama jurusan dari *record* dosen.

Dalam SQL, untuk menjawab kueri di atas, tabel yang perlu diakses dapat dicantumkan dengan *from*, dan menentukan kondisi yang cocok denngan *where*. Permintaan di atas dapat ditulis dalam SQL sebagai berikut:

```
    select                    nama,
dosen.nama_jurusan, lokasi
    from dosen, jurusan
    where
    dosen.nama_jurusan            =
    jurusan.nama_jurusan;
```

Tabel 3.4. Hasil Kueri Multi Tabel

Nama	Nama_Jurusan	Lokasi
Suparni	Ilmu Komputer	Kampus I
Andri	Sistem Informasi	Kampus I
Reza	Biologi	Kampus I
Bobby	Sistem Informasi	Kampus I
Sulastri	Matematika	Kampus I

Perhatikan bahwa atribut jurusan terdapat di tabel dosen dan jurusan, dan nama tabel digunakan sebagai awalan (dalam dosen.nama_jurusan, dan jurusan.nama_jurusan) untuk memperjelas atribut mana yang dimaksud. Sebaliknya, atribut nama dan lokasi hanya muncul di salah satu tabel, dan karenanya tidak perlu diawali dengan nama tabel.

Penamaan ini mensyaratkan bahwa tabel yang ada di dalam *from* memiliki nama yang berbeda. Persyaratan ini menyebabkan masalah dalam beberapa kasus, seperti ketika informasi dari dua *record* berbeda dalam tabel yang sama perlu digabungkan.

Seperti yang telah dipelajari sebelumnya, kueri SQL dapat berisi tiga jenis klausa, klausa *select, from* dan *where*. Peran setiap klausa adalah sebagai berikut:

- *Select* digunakan untuk mendaftar atribut yang diinginkan dalam hasil permintaan.
- *From* adalah daftar tabel yang akan diakses dalam evaluasi kueri.

- *Where* adalah predikat yang melibatkan atribut tabel dalam dari klausa *from*.

Meskipun klausa harus ditulis dalam urutan *select, from, where*, cara termudah untuk memahami operasi yang ditentukan oleh permintaan adalah dengan mempertimbangkan klausa dalam urutan operasional: pertama *from*, lalu *where*, dan kemudian *select*.

Tabel hasil memiliki semua atribut dari semua tabel dalam klausa *from*. Karena nama atribut yang sama dapat muncul pada lebih dari satu tabel, seperti yang dilihat sebelumnya, awali nama relasi dari mana atribut itu berasal, sebelum nama atribut.

Sebagai contoh, skema hubungan untuk tabel dosen dan kelas adalah:

```
(dosen.ID,          dosen.name,
dosen.nama_jurusan,
dosen.gaji, pengajaran.ID,
pengajaran.kode_MK,
pengajaran.kode_kelas,
pengajaran.semester,
pengajaran.tahun)
```

Dengan skema ini, dosen.ID dapat dibedakan dari pengajaran.ID. Untuk atribut yang hanya muncul di salah satu dari dua tabel, biasanya akan dibuang awalan nama-tabel. Penyederhanaan ini tidak mengarah pada ambiguitas apa pun. Skema hubungan dapat ditulis sebagai berikut:

```
    (dosen.        ID,        nama,
nama_jurusan, gaji
    pengajaran.ID,        kode_MK,
    kode_kelas, semester, tahun)


    select nama, kode_MK from
dosen, pengajaran
    where                dosen.ID=
pengajaran.ID;
```

Kueri di atas hanya menghasilkan dosen yang telah mengajar beberapa mata kuliah. Dosen yang belum mengajar mata kuliah apa pun bukanlah *output*, jika ingin menghasilkan *record* seperti itu, bisa digunakan operasi yang disebut *outer join*, yang akan dijelaskan berikutnya.

Jika hanya ingin menemukan nama dosen dan pengidentifikasi mata kuliah untuk dosen di jurusan Ilmu Komputer, dapat ditambahkan predikat tambahan untuk klausa *where*, seperti yang ditunjukkan di bawah ini.

```
    select nama, kode_MK from
dosen, pengajaran
    where        dosen.ID        =
    pengajaran.ID              and
    dosen.nama_jurusan   =   'Ilmu
    Komputer';
```

3.4. Natural Join

Sebelumnya dalam kueri contoh yang menggabungkan informasi dari tabel dosen dan

tabel pengajaran, kondisi yang cocok mengharuskan dosen.ID sama dengan pengajaran.ID. Ini adalah satu-satunya atribut dalam dua tabel yang memiliki nama yang sama. Sebenarnya ini adalah kasus umum; yaitu, kondisi pencocokan dalam klausa *from* paling sering mengharuskan semua atribut dengan nama yang cocok untuk disamakan.

Untuk membuat kehidupan seorang *programmer* SQL lebih mudah untuk kasus umum ini, SQL mendukung operasi yang disebut *natural join*. Bahkan SQL mendukung beberapa cara lain di mana informasi dari dua atau lebih tabel dapat digabungkan bersama. Telah terlihat bagaimana predikat klausa dapat digunakan untuk menggabungkan informasi dari banyak tabel.

Operasi gabungan alami beroperasi pada dua tabel dan menghasilkan tabel lain sebagai hasilnya. *Natural join* menganggap hanya pasangan *record* dengan nilai yang sama pada atribut yang muncul dalam skema kedua tabel. Jadi, kembali ke contoh hubungan dosen dan pengajaran, dosen bergabung dengan pengajaran di mana *record* dari dosen dan pengajaran memiliki nilai yang sama pada atribut umum, yaitu ID.

Pertimbangkan pertanyaan “Untuk semua dosen di universitas yang telah mengajar beberapa mata kuliah, temukan nama mereka dan kode mata kuliah dari semua program yang mereka ajarkan”, yang ditulis sebelumnya sebagai:

```
select nama, kode_MK
from dosen, pengajaran
where dosen.ID = pengajaran.ID;
```

Query ini dapat ditulis lebih ringkas menggunakan operasi natural-join di SQL sebagai:

```
select nama, kode_MK
from dosen natural join pengajaran;
```

Kedua kueri di atas menghasilkan hasil yang sama. Seperti yang dilihat sebelumnya, hasil dari operasi natural join adalah tabel. Secara konseptual, ungkapan "dosen *natural join* pengajaran" pada klausa digantikan oleh hubungan yang diperoleh dengan menilai *natural join*. Klausa *where* dan *select* kemudian dievaluasi pada relasi ini.

Sebagai contoh, misalkan ingin menjawab pertanyaan "Daftar nama dosen bersama dengan judul mata kuliah yang mereka ajarkan." Permintaan dapat ditulis dalam SQL sebagai berikut:

```
select nama, judul
from dosen natural join pengajaran, mata_kuliah
where pengajaran.kode_MK = mata_kuliah.kode_MK;
```

3.5. Operasi Dasar Tambahan

3.5.1. Operasi *Rename*

Perhatikan kembali kueri yang telah digunakan sebelumnya:

```
    select nama, kode_MK
    from dosen, pengajaran
    where       dosen.ID        =
pengajaran.ID;
```

Hasil kueri ini adalah tabel dengan atribut nama dan kode mata kuliah saja. Nama-nama atribut dalam hasil berasal dari nama-nama atribut dalam tabel dari klausa *from*.

Namun, pengguna tidak dapat selalu mendapatkan nama dengan cara ini, karena beberapa alasan:

- Dua relasi dalam klausa *from* dapat memiliki atribut dengan nama yang sama, dalam hal ini nama atribut diduplikasi dalam hasilnya.
- Jika pengguna menggunakan ekspresi aritmatika dalam klausa *select*, atribut yang dihasilkan tidak memiliki nama.
- Jika nama atribut dapat diturunkan dari tabel dasar seperti pada contoh sebelumnya, penguuna mungkin ingin mengubah nama atribut dalam hasilnya.

Oleh karena itu, SQL menyediakan cara mengubah nama atribut dari tabel hasil, yakni dengan menggunakan klausa *as*, yang dapat dituliskan sebagai berikut:

```
    select nama as nama dosen,
kode_MK
    from dosen, pengajaran
    where                dosen.ID=
pengajaran.ID;
```

Klausa *as* sangat berguna dalam mengubah nama tabel. Salah satu alasan untuk mengganti nama tabel adalah mengganti nama tabel yang panjang dengan versi singkat yang lebih nyaman digunakan di tempat lain dalam kueri. Sebagai ilustrasi, ditulis ulang kueri "Untuk semua dosen di universitas yang telah mengajar beberapa mata kuliah, temukan nama mereka dan kode mata kuliah dari semua program yang mereka ajarkan."

```
    select T.nama, S.kode_MK
    from dosen as T, pengajaran
as S
    where T.ID= S.ID;
```

Alasan lain untuk mengganti nama tabel adalah kasus di mana ingin membandingkan *record* dalam tabel yang sama. Misalkan ingin menulis kueri "Temukan nama semua dosen yang gajinya lebih besar dari setidaknya satu dosen di jurusan Sistem Informasi." Dapat ditulis ekspresi SQL:

```
    select distinct T.nama
    from dosen as T, dosen as S
    where T.gaji > S.gaji and
    S.nama_jurusan = 'Sistem
    Informasi';
```

Dalam kueri di atas, T dan S dapat dianggap sebagai salinan tabel dosen, tetapi lebih tepatnya, mereka dinyatakan sebagai alias, yaitu sebagai nama alternatif, untuk tabel dosen. *Identifier*, seperti T dan S, yang digunakan untuk mengubah nama tabel disebut sebagai nama korelasi dalam standar SQL, tetapi juga biasa disebut sebagai tabel alias, atau variabel korelasi, atau variabel *tuple*/*record*.

3.5.2. Operasi *String*

SQL menentukan *string* dengan melampirkannya dalam tanda kutip tunggal, misalnya, 'Komputer'. Karakter kutipan tunggal yang merupakan bagian dari *string* dapat ditentukan dengan menggunakan dua karakter kutipan tunggal; misalnya, *string* "it's me" dapat diganti dengan "It''s me". Standar SQL menetapkan bahwa operasi kesetaraan pada *string* adalah *case-sensitive*, akibatnya ungkapan "' comp. sci. '= 'Comp. Sci. '" akan bernilai *false*.

Namun, beberapa sistem basis data, seperti *MySQL* dan *SQL Server*, tidak membedakan huruf besar dari huruf kecil saat mencocokkan *string*; sebagai hasilnya "' comp. sci. '= 'Comp. Sci. '"akan bernilai *true* pada *database* ini. Namun perilaku *default* ini dapat diubah, baik di tingkat basis data atau di tingkat atribut tertentu.

SQL juga memungkinkan berbagai fungsi pada karakter *string*, seperti menggabungkan (menggunakan "||"), Mengekstraksi *substring*,

menemukan panjang *string*, mengubah *string* menjadi huruf besar (menggunakan fungsi *upper*(s) di mana s adalah *string*) dan huruf kecil (menggunakan fungsi *lower*(s)), menghilangkan spasi di akhir *string* (menggunakan *trim*(s)) dan sebagainya. Ada variasi pada rangkaian fungsi *string* yang didukung oleh sistem basis data yang berbeda.

Pencocokan pola dapat dilakukan pada *string*, menggunakan operator *like*. Digambarkan pola dengan menggunakan dua karakter khusus:

- Persen (%): karakter % cocok dengan semua *substring*.
- Garis Bawah (_): Karakter tersebut cocok dengan karakter apa pun.

Pola sensitif terhadap huruf besar-kecil; artinya, huruf besar tidak cocok dengan huruf kecil, atau sebaliknya. Untuk menggambarkan pencocokan pola, pertimbangkan contoh berikut:

- ’Intro%’ cocok dengan sembarang *string* yang dimulai dengan “Intro”
- ’%Komp%’ cocok dengan *string* apa pun yang mengandung "Komp" sebagai *substring*, misalnya, 'Ilmu Komputer', dan 'Biologi Komputasi'
- ’_ _ _’ cocok dengan *string* apa pun yang persis tiga karakter.
- ’_ _ _%’ cocok dengan *string* apa pun yang setidaknya tiga karakter.

SQL mengekspresikan pola dengan menggunakan operator perbandingan sejenis. Pertimbangkan kueri “Temukan nama semua

jurusan yang lokasinya mencakup *substring* ‘ Kampus ’.” Kueri ini dapat ditulis sebagai:

```
select nama_jurusan
from jurusan
where lokasi like '%Kampus%';
```

Agar pola menyertakan karakter pola khusus (yaitu,% dan _), SQL memungkinkan spesifikasi karakter *escape*. Karakter *escape* digunakan segera sebelum karakter pola khusus untuk menunjukkan bahwa karakter pola khusus harus diperlakukan seperti karakter normal. Didefinisikan karakter *escape* untuk perbandingan serupa menggunakan garis miring terbalik (\).

- like ’ab\%cd%’ *escape* ’\’ cocok dengan semua string yang dimulai dengan “ab%cd”.
- like ’ab\\cd%’ *escape* ’\’ cocok dengan semua string yang dimulai dengan “ab\cd”.

3.5.3. Spesifikasi Atribut pada Klausa *Select*

```
select dosen.*
from dosen, pengajaran
where dosen.ID = pengajaran.ID;
```

Simbol tanda bintang "*" dapat digunakan dalam klausa *select* untuk menunjukkan "semua atribut". Dengan demikian, penggunaan dosen.* dalam klausa *select* menunjukkan bahwa semua atribut dosen harus

dipilih. Klausa *select* * menunjukkan bahwa semua atribut dari tabel hasil dari klausa *from* dipilih.

3.5.4. Menyusun Tampilan *Record*

SQL menawarkan kepada pengguna kontrol atas urutan *record* dalam suatu tabel yang ditampilkan. Klausa *order by* menyebabkan *record* dalam hasil kueri muncul dalam urutan yang diurutkan. Untuk mendaftar dalam urutan abjad semua dosen di jurusan sistem informasi, kuerinya adalah:

```
    select nama
    from dosen
    where    jurusan=    'Sistem
Informasi'
    order by nama;
```

Secara default, klausa *order by* mencantumkan item dalam urutan menaik. Untuk menentukan urutan pengurutan, dapat ditentukan *desc* untuk urutan menurun atau *asc* untuk urutan naik. Selanjutnya, penyusunan dapat dilakukan pada beberapa atribut. Misalkan pengguna ingin membuat daftar seluruh tabel dosen dalam urutan gaji yang menurun. Jika beberapa dosen memiliki gaji yang sama, pengguna menyusunnya dalam urutan berdasarkan nama. Dinyatakan kueri ini dalam SQL sebagai berikut:

```
    select *
    from dosen
    order by gaji desc, nama asc;
```

3.5.5. Predikan Klausa *Where*

SQL mencakup operator perbandingan *between* untuk menyederhanakan klausa *where* yang menentukan bahwa nilai lebih kecil atau sama dengan beberapa nilai dan lebih besar dari atau sama dengan beberapa nilai lainnya. Jika pengguna ingin menemukan nama dosen dengan jumlah gaji antara 4.500.000 dan 5.500.000, pengguna dapat menggunakan perbandingan *between* dengan kueri sebagai berikut:

```
    select nama
    from dosen
    where gaji between 3500000
and 4500000;
```

dari pada:

```
    select nama
    from dosen
    where gaji<= 4500000 and gaji
>= 3500000;
```

Selain operator perbandingan *between*, pengguna juga dapat menggunakan operator perbandingan *not between*.

3.6. Operasi *Set*

Operasi SQL, *union*, *intersect*, dan *except* beroperasi pada tabel dan sesuai dengan operasi teori himpunan matematika ∪, ∩, dan -.

```
    select kode_MK
    from kelas
    where semester = 1 and tahun
= 2017;
```

Kueri di atas akan menghasilkan tabel yang berisi semua kode mata kuliah untuk kelas semester 1 tahun 2017.

Tabel 3.5. Mata Kuliah Semester 1 tahun 2017

Kode_MK
AB-101
AB-287
CH-156

```
    select kode_MK
    from kelas
    where semester = 2 and tahun
= 2018;
```

Kueri di atas akan menghasilkan tabel yang berisi semua kode mata kuliah untuk kelas semester 2 tahun 2018.

Tabel 3.6. Mata Kuliah Semester 2 tahun 2018

Kode_MK
AB-101
DN-206
MI-213

MI-213
KS-119
TS-287
SA-212

3.6.1. Union

Untuk menemukan rangkaian semua mata kuliah yang diajarkan pada semester 1 atau semester 2, atau keduanya, kuerinya:

```
    (select kode_MK
    from kelas
    where semester = 1 and tahun=
2017)
    union
    (select kode_MK
    from kelas
    where semester = 2 and tahun=
2018);
```

Operasi *union* secara otomatis menghilangkan duplikat, tidak seperti klausa *select*. Dengan demikian, tabel dua bagian mata kuliah yang ditawarkan pada semester 1 dan semester 2 akan muncul pada tabel hasil, dan *record* yang sama hanya muncul sekali dalam hasil.

Tabel 3.7. Tabel Hasil *union*

Kode_MK

AB-101
AB-287
CH-156
DN-206
MI-213
KS-119
TS-287
SA-212

Jika ingin mempertahankan semua duplikat, harus ditulis *union all* sebagai ganti *union*:

```
    (select kode_MK
    from kelas
    where semester = 1 and tahun=
2017)
    union all
    (select kode_MK
    from kelas
    where semester = 2 and tahun=
2018);
```

Tabel 3.8. Tabel Hasil *union all*

Kode_MK
AB-101
AB-287
CH-156
AB-101
DN-206
MI-213
MI-213
KS-119

TS-287
SA-212

3.6.2. Intersect

Untuk menemukan rangkaian semua mata kuliah yang diajarkan pada semester 1 2017 dan juga pada semester 2 tahun 2018, kuerinya:

```
    (select kode_MK
    from kelas
    where semester = 1 and tahun=
2017)
    intersect
    (select kode_MK
    from kelas
    where semester = 2 and tahun=
2018);
```

Tabel 3.9. Tabel Hasil *intersect*

Kode_MK
AB-101

Tabel hasil, hanya berisi *record* yang yang terdapat pada klausa *select* pertama dan juga terdapat pada klusa *select* yang kedua. Operasi *intersect* secara otomatis menghilangkan duplikat. Sebagai contoh, jika EC-101 diajarkan 2 kali pada semester 1 tahun 2017 dan 2 kali pada semester 2 tahun 2018, maka hanya akan ada 1 *record* dengan EC-101 di tabel hasil. Jika ingin mempertahankan semua duplikat, harus ditulis *intersect all* sebagai ganti *intersect*:

```
    (select kode_MK
    from kelas
```

```
    where semester = 1 and tahun=
2017)
    intersect all
    (select kode_MK
    from kelas
    where semester = 2 and tahun=
2018);
```

Tabel 3.10. Tabel Hasil *intersect all*

Kode_MK
AB-101

Jumlah duplikat *record* yang muncul dalam hasilnya sama dengan jumlah minimum duplikat di tabel hasil *intersect*. Namun, jika EC-101 diajarkan 2 kali pada semester 1 tahun 2017 dan 2 kali pada semester 2 tahun 2018, maka akan ada 2 *record* dengan EC-101 di tabel hasil.

3.6.3. Except

Untuk menemukan semua mata kuliah yang diajarkan pada semester 1 tahun 2017 tetapi tidak pada semester 2 tahun 2018, kuerinya:

```
    (select kode_MK
    from kelas
    where semester = 1 and tahun=
2017)
    except
    (select kode_MK
    from kelas
    where semester = 2 and tahun=
2018);
```

Tabel 3.11. Tabel Hasil *except*

Kode_MK
AB-287
CH-156

Tabel hasil menunjukkan isi dari tabel mata kuliah semseter 1 tahun 2017 kecuali untuk AB-101 tidak muncul. Operasi *except*, mengeluarkan semua *record* dari input pertama yang tidak terdapat pada input kedua. Operasi *except* secara otomatis menghilangkan duplikat dalam input sebelum melakukan perbandingan. Misalnya, jika EC-101 diajarkan 2 kali pada semester 1 tahun 2017 dan 2 kali pada semester 2 tahun 2018, hasil operasi *except* tidak akan memiliki salinan EC-101. Jika ingin menyimpan duplikat, harus ditulis *except all* sebagai pengganti *except*:

```
    (select kode_MK
    from kelas
    where semester = 1 and tahun=
2017)
    except all
    (select kode_MK
    from kelas
    where semester = 2 and tahun=
2018);
```

3.7. Null

Nilai *null* memberikan masalah khusus dalam operasi relasional, termasuk operasi aritmatika, operasi perbandingan, dan operasi yang ditetapkan. Hasil dari ekspresi aritmatika (yang melibatkan, misalnya +, -, ∗, atau /) adalah *null* jika ada nilai input yang *null*. Misalnya, jika kueri memiliki ekspresi r + 5, dan r adalah *null* untuk *record* tertentu, maka hasil ekspresi juga harus *null* untuk *record* itu.

Perbandingan yang melibatkan *null* lebih merupakan masalah. Sebagai contoh, pertimbangkan perbandingan "1 < *null*". Akan salah untuk mengatakan ini benar karena tidak tahu apa yang diwakili nilai *null*. Tetapi juga akan salah untuk mengklaim ungkapan ini salah; jika dilakukan, "*not* (1 < *null*)" akan mengevaluasi ke true, yang tidak masuk akal. Oleh karena itu SQL memperlakukan sebagai tidak diketahui (*unknown*) hasil perbandingan yang melibatkan nilai *null* (selain predikat *is null* dan *is not null*, yang akan dijelaskan berikutnya). Ini menciptakan nilai logis ketiga selain benar dan salah. Karena predikat klausa *where* dapat melibatkan operasi *Boolean* seperti *and*, *or*, dan *not* pada hasil perbandingan, definisi operasi *Boolean* diperluas untuk menangani nilai yang *unknown*.

Tabel 3.12. Perbandingan Nilai *Unknown*

Kondisi 1	Operator	Kondisi 2	Hasil
true	**and**	unknown	unknown
false		unknown	false

unknown		unknown	unknown
true	**or**	unknown	true
false		unknown	unknown
unknown		unknown	unknown
	not	unknown	unknown

Jika predikat klausa *where* mengevaluasi salah atau tidak dikenal untuk *record*, *record* itu tidak ditambahkan ke hasilnya. SQL menggunakan kata kunci khusus *null* dalam predikat untuk menguji nilai *null*. Dengan demikian, untuk menemukan semua dosen yang muncul dalam tabel dosen dengan nilai *null* untuk gaji, kuerinya:

```
select nama
from dosen
where gaji is null;
```

Predikatnya *is not null* berhasil jika nilai yang diterapkannya bukan *null*. Beberapa implementasi SQL juga memungkinkan untuk menguji apakah hasil perbandingan tidak diketahui (*unknown*), bukan benar (*true*) atau salah (*false*), dengan menggunakan klausa *is unknown* dan *is not unknown*. Ketika kueri menggunakan klausa *select distinct*, duplikat *record* harus dihilangkan. Untuk tujuan ini, ketika membandingkan nilai atribut yang sesuai dari dua *record*, nilai tersebut diperlakukan sama jika keduanya *non-null* dan nilainya sama, atau keduanya *null*. Dengan demikian dua salinan *record*, seperti {('A', *null*), ('A', *null*)}, diperlakukan sebagai identik, walaupun beberapa

atribut memiliki nilai *null*. Dengan menggunakan klausa *distinct* maka hanya menyimpan satu salinan *record* identik tersebut. Perhatikan bahwa perlakuan *null* di atas berbeda dari cara *null* diperlakukan dalam predikat, di mana perbandingan "*null = null*" akan mengembalikan nilai yang tidak diketahui, daripada benar. Pendekatan di atas memperlakukan *record* sebagai identik jika mereka memiliki nilai yang sama untuk semua atribut, bahkan jika beberapa nilai adalah *null*, juga digunakan untuk kesatuan operasi *union*, *intersection* dan *except*[5].

3.8. Modifikasi Basis Data

Berikut akan dijelaskan cara menambah, menghapus, atau mengubah informasi dengan SQL.

3.8.1. Delete

Permintaan penghapusan diungkapkan dengan cara yang hampir sama dengan kueri. Pengguna hanya dapat menghapus seluruh baris data, dan tidak dapat menghapus nilai hanya pada atribut tertentu. SQL menyatakan penghapusan dengan kueri:

```
delete from r
where P;
```

Dimana P mewakili predikat dan r mewakili suatu tabel. Pernyataan hapus pertama-

[5] Raghu Ramakrishnan dan Johannes Gehkre, *Database Management Systems Third Edition* (Mc Graw Hill, 2003).

tama menemukan semua *record* t dalam r yang P (t) benar, lalu menghapusnya dari r. Klausa *where* dapat dihilangkan, dalam hal ini semua *record* dalam r akan dihapus.

Perhatikan bahwa perintah hapus hanya beroperasi pada satu tabel. Jika ingin menghapus *record* dari beberapa tabel, harus menggunakan satu perintah delete untuk setiap tabel. Perintah ini menghapus semua *record* dari tabel dosen, tabel dosen itu sendiri masih ada, tetapi kosong.

Berikut adalah contoh permintaan penghapusan SQL

- Hapus semua *record* dalam tabel dosen yang berkaitan dengan dosen di jurusan biologi.

```
delete from dosen
where nama_jurusan= 'biologi';
```

- Hapus semua dosen dengan gaji antara 1.000.000 dan 2.000.000.

```
delete from dosen
where gaji between 1000000 and 2000000;
```

- Hapus semua *record* dalam tabel dosen untuk dosen yang terkait dengannya sebuah jurusan yang terletak di Kampus I.

```
delete from dosen
where nama_jurusan in (select nama_jurusan
from jurusan
where lokasi= 'Kampus I');
```

Permintaan penghapusan ini pertama-tama menemukan semua nama jurusan yang berlokasi di Kampus I, dan kemudian menghapus semua *record* dosen yang berkaitan dengan jurusan tersebut.

Perhatikan bahwa, meskipun pengguna dapat menghapus *record* dari hanya satu tabel pada satu waktu, pengguna dapat mereferensikan sejumlah relasi mana pun di dalam *select-from-where* di tempat penghapusan. Permintaan penghapusan dapat berisi seleksi bersarang yang mereferensikan tabel dari mana *record* akan dihapus. Misalnya, jika ingin menghapus catatan semua dosen dengan gaji di bawah rata-rata di universitas. Kuerinya dapat ditulis sebagai berikut:

```
    delete from dosen
    where   gaji<   (select   avg
(gaji)
    from dosen);
```

3.8.2. Insert

Untuk memasukkan data ke dalam tabel, ditentukan *record* yang akan disisipkan atau menulis kueri yang hasilnya adalah serangkaian *record* yang akan disisipkan. Jelas, nilai atribut untuk *record* yang dimasukkan harus anggota dari domain atribut yang sesuai. Demikian pula, *record* yang dimasukkan harus memiliki jumlah atribut yang benar.

Pernyataan *insert* paling sederhana adalah permintaan untuk memasukkan satu *record*.

Misalkan ingin memasukkan fakta bahwa ada mata kuliah BD-123 di jurusan Sistem Informasi dengan judul "Sistem Basis Data", dengan jumlah 4 sks. Kuerinya adalah:

```
    insert into mata_kuliah
    values   ('BD-123',   'Sistem
Basis Data', 'Sistem Informasi',
4);
```

Dalam contoh ini, nilai ditentukan dalam urutan di mana atribut yang sesuai tercantum dalam skema tabel. Untuk kepentingan pengguna yang mungkin tidak ingat urutan atribut, SQL memungkinkan atribut ditentukan sebagai bagian dari pernyataan *insert*. Misalnya, pernyataan *insert* SQL berikut ini fungsinya identik dengan yang sebelumnya:

```
    insert    into    mata_kuliah
(kode_MK,  judul,  nama_jurusan,
sks)
    values   ('BD-123',   'Sistem
Basis Data', 'Sistem Informasi',
4);
```

atau

```
    insert    into    mata_kuliah
(judul,       kode_MK,       sks,
nama_jurusan)
    values ('Sistem Basis Data',
'BD-123', 4, 'Sistem Informasi');
```

Secara umum, pengguna mungkin ingin menyisipkan *record* berdasarkan hasil dari kueri. Misalkan ingin membuat setiap mahasiswa di jurusan biologi yang telah memperoleh lebih dari 144 sks, menjadi seorang dosen di jurusan biologi, dengan gaji 2.000.000. Kuerinya:

```
    insert into dosen
    select        ID,        nama,
nama_jurusan, 2000000
    from mahasiswa
    where nama_jurusan= 'Biologi'
and total_sks> 144;
```

SQL mengevaluasi pernyataan *select* terlebih dahulu, memberikan satu set *record* yang kemudian dimasukkan ke dalam tabel dosen. Setiap *record* memiliki ID, nama, nama_jurusan, dan gaji.

3.8.3. Update

Dalam situasi tertentu, pengguna mungkin ingin mengubah nilai dalam sebuah *record* tanpa mengubah semua nilai dalam *record*. Untuk tujuan ini, pernyataan *update* dapat digunakan. Seperti memasukkan dan menghapus, pengguna juga dapat memilih *record* untuk diubah dengan menggunakan kueri.

Misalkan kenaikan gaji tahunan sedang dilakukan, dan gaji semua dosen akan dinaikkan sebesar 5 persen. Kuerinya:

```
    update dosen
```

```
set gaji= gaji* 1.05;
```

Pernyataan *update* sebelumnya diterapkan sekali untuk masing-masing *record* dalam tabel dosen.

Jika kenaikan gaji hanya dibayarkan kepada dosen dengan gaji kurang dari 2.000.000, kuerinya:

```
update dosen
set gaji= gaji* 1.05
where gaji< 2000000;
```

Seperti sebelumnya, SQL terlebih dahulu menguji semua *record* dalam tabel untuk melihat apakah mereka harus diperbarui, dan melakukan pembaruan sesudahnya.

Contoh lain, pengguna dapat menulis permintaan "Berikan kenaikan gaji 5 persen kepada dosen yang gajinya kurang dari rata-rata" sebagai berikut:

```
update dosen
set gaji= gaji* 1.05
where gaji < (select avg (gaji)
from dosen);
```

3.9. Latihan

1. Jelaskan tipe-tipe dasar variabel SQL.
2. Tuliskan kueri SQL untuk menampilkan nama dosen yang hanya pada jurusan sistem informasi saja.

3. Tuliskan kueri SQL untuk menemukan rangkaian semua mata kuliah yang diajarkan pada semester 1 2017 dan juga pada semester 2 tahun 2018.

BAB IV

Entity Relationship (ER)

4.1. Entity Relationship Model

Model data *entity relationship* (ER) memungkinkan untuk menggambarkan data yang terlibat dalam perusahaan dunia nyata dalam hal objek dan hubungannya dan secara luas digunakan untuk mengembangkan desain basis data awal. Ini memberikan konsep-konsep berguna yang memungkinkan untuk beralih dari deskripsi informal tentang apa yang diinginkan pengguna dari database mereka ke deskripsi yang lebih rinci dan tepat yang dapat diimplementasikan dalam DBMS.

Dalam *entity relationship model* (atau model E / R), struktur data direpresentasikan secara grafis, sebagai "*Entity Relationship Diagram (ERD)*" menggunakan tiga prinsip jenis elemen yaitu set entitas, atribut, dan relasi(hubungan).

Model ER sangat berguna dalam memetakan makna dan interaksi perusahaan dunia nyata ke dalam skema konseptual. Karena khasiatnya ini, banyak alat desain database menggunakan konsep dari model ER . Model data ER menggunakan tiga konsep dasar: set entitas, set relasi, dan atribut.

4.2. Entitas

Entitas adalah "benda" atau "objek" di dunia nyata yang dapat dibedakan dari semua

objek lainnya. Misalnya, setiap orang di universitas adalah entitas. Entitas memiliki seperangkat properti, dan nilai untuk beberapa set properti mungkin secara unik mengidentifikasi entitas. Misalnya, seseorang mungkin memiliki properti id yang nilainya unik yang saya mengidentifikasikan orang tersebut. Dengan demikian, nilai 192110761 untuk id orang akan secara unik mengidentifikasi satu orang tertentu di universitas. Demikian pula, mata kuliah dapat dianggap sebagai entitas, dan kode mata kuliah secara unik mengidentifikasi entitas mata kuliah di universitas. Suatu entitas mungkin konkret, seperti orang atau buku, atau mungkin abstrak, seperti mata kuliah, perkuliahan, atau penjualan.

Entity set (set entitas) adalah himpunan entitas dengan tipe yang sama yang berbagi properti atau atribut yang sama. Himpunan semua orang yang menjadi dosen di universitas tertentu, misalnya, dapat didefinisikan sebagai set entitas dosen. Demikian pula, himpunan entitas mahasiswa dapat mewakili himpunan semua mahasiswa di universitas.

Dalam proses pemodelan, sering digunakan istilah entitas yang ditetapkan dalam abstrak, tanpa merujuk ke set entitas individu tertentu. Digunkan perpanjangan istilah entitas yang ditetapkan untuk merujuk pada koleksi entitas yang sebenarnya milik entitas yang ditetapkan. Dengan demikian, himpunan dosen di universitas membentuk perpanjangan dari set entitas dosen.

Set entitas tidak perlu dipisahkan. Sebagai contoh, adalah mungkin untuk mendefinisikan set entitas dari semua orang di universitas. Entitas seseorang dapat berupa entitas dosen, entitas mahasiswa, keduanya, atau tidak sama sekali.

Entitas diwakili oleh seperangkat atribut. Atribut adalah sifat deskriptif yang dimiliki oleh setiap anggota entitas yang ditetapkan. Penunjukan atribut untuk set entitas menyatakan bahwa database menyimpan informasi yang sama mengenai setiap entitas dalam set entitas. Namun, setiap entitas dapat memiliki nilai sendiri untuk setiap atribut. Atribut yang mungkin dari kumpulan entitas dosen adalah ID, nama, nama jurusan, dan gaji. Atribut yang mungkin dari set entitas mata kuliah adalah kode_MK, judul, nama jurusan, dan sks.

Setiap entitas memiliki nilai untuk setiap atributnya. Misalnya, entitas dosen tertentu dapat memiliki nilai 10021 untuk ID, nilai Raissa untuk nama, nilai Sistem Informasi untuk nama jurusan, dan nilai 3000000 untuk gaji.

Atribut ID digunakan untuk mengidentifikasi dosen secara unik, karena mungkin ada lebih dari satu dosen dengan nama yang sama. Di Indonesia, NIK(Nomor Induk Kependudukan) digunakan sebagai atribut yang nilainya mengidentifikasi orang tersebut secara unik. Secara umum perusahaan harus membuat dan menetapkan pengidentifikasi unik untuk setiap karyawannya.

Basis data mencakup kumpulan set entitas, yang masing-masing berisi sejumlah entitas dari jenis yang sama. Tabel 4.1 dan 4.2 menunjukkan bagian dari basis data universitas yang terdiri dari dua set entitas: dosen dan mahasiswa. Agar gambarnya sederhana, hanya beberapa atribut dari dua set entitas yang ditampilkan.

Tabel 4.1. Set Entitas Dosen

10021	Raissa
10024	Laylan
10031	Franindya
10049	Sriani

Tabel 4.2. Set Entitas Mahasiswa

192110761	Sri Wahyuni
192122434	Indah Lestari
193213423	Tommy
180898392	Antonio
182379002	Yuli Astari

Basis data untuk universitas dapat mencakup sejumlah set entitas lainnya. Sebagai contoh, selain melacak dosen dan mahasiswa, universitas juga memiliki informasi tentang mata kuliah, yang diwakili oleh entitas yang mengatur mata kuliah dengan atribut kode_MK, judul, nama jurusan dan sks. Dalam lingkungan nyata, basis data universitas dapat menyimpan puluhan set entitas.

4.2.1. Entitas Kuat dan Entitas Lemah

Contoh sebelumnya menggambarkan entitas kuat, yaitu entitas mandiri yang keberadaannya tidak bergantung kepada keberadaan entitas yang lainnya. Sebaliknya, entitas lemah adalah entitas yang keberadaannya bergantung kepada keberadaan entitas lainnya. Entitas lemah tidak akan memiliki arti di dalam ERD tanpa adanya entitas kuat lainnya tempat entitas tersebut bergantung. Entitas lemah tidak memiliki pengidentifikasinya sendiri dan berperan sebagai pengidentifikasi sebagian. Sebagai contoh, entitas pasangan dosen yang merupakan entitas lemah yang bergantung pada entitas dosen. Tanpa adanya entitas dosen, maka entitas pasangan dosen tidak memiliki arti dan tidak dibutuhkan.

4.2.2. Entitas Asosiatif

Entitas asosiatif adalah entitas yang terbentuk dari suatu relasi dan tidak bisa berdiri sendiri. Entitas Asosiatif digambarkan dengan kotak persegi panjang dengan belah ketupat di bagian dalamnya. Sebagai contoh entitas ijazah yang terbentuk antara entitas mahasiswa dengan entitas kuliah dengan relasi mengambil. Relasi yang sebenarnya adalah mahasiswa mengambil kuliah, kemudian pada akhir perkuliahan mahasiswa akan mendapatkan ijazah. Oleh karena itu terbentuk entitas asosiatif ijazah yang mana hanya dapat muncul setelah mahasiswa menyelesaikan kuliah, namun jika mahasiswa tersebut tidak memenuhi persyaratan

menyelesaikan kuliah, maka mahasiswa tersebut tidak mendapatkan ijazah[6].

4.3. Atribut

Untuk setiap atribut, ada satu set nilai yang diizinkan, yang disebut domain, atau set nilai, dari atribut itu. Domain dari atribut kode mata kuliah mungkin himpunan semua string teks dengan panjang tertentu. Demikian pula, domain atribut semester mungkin berupa string dari set {genap, ganjil}.

Secara formal, atribut dari set entitas adalah fungsi yang memetakan dari entitas yang ditetapkan ke domain. Karena himpunan entitas mungkin memiliki beberapa atribut, masing-masing entitas dapat dijelaskan oleh satu set pasangan (atribut, nilai data), satu pasang untuk setiap atribut dari kumpulan entitas. Misalnya, entitas dosen tertentu dapat dijelaskan oleh set {(ID, 10021), (nama, Raissa), (nama jurusan, Sistem Informasi), (gaji, 3000000)}, yang berarti entitas menggambarkan seseorang bernama Raissa yang ID dosennya adalah 10021, yang merupakan anggota jurusan Sistem Informasi dengan gaji Rp. 3.000.000. Nilai atribut yang menggambarkan suatu entitas merupakan bagian yang signifikan dari data yang disimpan dalam database.

[6] Adi Nugroho, *Perancangan dan Implementasi Sistem Basis Data* (Penerbit Andi, 2011).

Atribut, seperti yang digunakan dalam model E-R, dapat dikarakterisasi dengan tipe atribut berikut:

4.3.1. Atribut Sederhana dan Komposit

Contoh – contoh sebelumnya merupakan atribut sederhana; yaitu, mereka belum dibagi menjadi beberapa bagian. Gabungan atribut, di sisi lain, dapat dibagi menjadi beberapa bagian. Misalnya, nama atribut dapat disusun sebagai atribut komposit yang terdiri dari nama depan, nama tengah, dan nama belakang. Menggunakan atribut komposit dalam skema desain adalah pilihan yang baik jika pengguna ingin merujuk keseluruh atribut pada beberapa kesempatan, dan hanya komponen atribut pada kesempatan lain. Misalkan harus menambahkan alamat ke entitas mahasiswa. Alamat dapat didefinisikan sebagai atribut alamat gabungan dengan atribut jalan, kota, negara bagian, dan kode pos. 3 Atribut komposit membantu mengelompokkan atribut terkait, membuat pemodelan lebih jelas. Perhatikan juga bahwa atribut gabungan dapat muncul sebagai hirarki. Dalam atribut komposit alamat, komponen jalan atributnya dapat dibagi lebih lanjut menjadi nomor jalan, nama jalan, dan nomor apartemen.

4.3.2. Atribut Bernilai Tunggal dan Multinilai

Atribut dalam contoh sebelumnya memiliki nilai tunggal untuk entitas tertentu. Misalnya, atribut ID dosen untuk entitas dosen tertentu

merujuk hanya satu ID dosen. Atribut tersebut dikatakan bernilai tunggal. Mungkin ada contoh di mana atribut memiliki aset nilai untuk entitas tertentu. Misalkan menambah set entitas dosen sebuah nomor atribut telepon. Seorang dosen mungkin memiliki nol, satu, atau beberapa nomor telepon, dan dosen yang berbeda mungkin memiliki nomor telepon yang berbeda. Atribut jenis ini dikatakan multinilai. Bila perlu, batas atas dan bawah dapat ditempatkan pada nomor telepon dalam atribut multinilai. Misalnya, universitas dapat membatasi jumlah nomor telepon yang dicatat untuk satu dosen tunggal menjadi dua. Penempatan batas dalam hal ini menyatakan bahwa atribut nomor telepon instruktur mungkin memiliki antara nol dan dua nilai.

4.3.3. Atribut yang diturunkan

Nilai untuk tipe atribut dapat diturunkan dari nilai atribut atau entitas terkait lainnya. Sebagai contoh, katakan bahwa set entitas dosen memiliki atribut yang membimbing mahasiswa, yang mewakili banyaknya mahasiswa yang dibimbing oleh dosen. Nilai untuk atribut ini dapat diperoleh dengan menghitung jumlah entitas mahasiswa yang terkait dengan dosen itu. Sebagai contoh lain, anggap bahwa kumpulan entitas dosen memiliki atribut usia yang menunjukkan usia dosen. Jika set entitas dosen juga memiliki atribut tanggal lahir, dapat dihitung usia dari tanggal lahir dan tanggal saat ini. Dengan demikian, usia adalah atribut turunan. Dalam hal ini, tanggal lahir dapat dirujuk sebagai

atribut dasar, atau atribut yang disimpan. Nilai atribut yang diturunkan tidak disimpan tetapi dihitung saat diperlukan.

Atribut mengambil nilai *null* ketika entitas tidak memiliki nilai. Nilai *null* juga dapat menunjukkan "tidak berlaku", yaitu bahwa nilai tersebut tidak ada untuk entitas. Misalnya, seseorang mungkin tidak memiliki nama tengah. *Null* juga bisa menetapkan bahwa nilai atribut tidak diketahui. Nilai yang tidak diketahui bisa karena hilang (nilainya ada, tetapi kami tidak memiliki informasi itu) atau tidak diketahui (tidak tahu apakah nilainya benar-benar ada). Misalnya, jika nilai nama untuk dosen tertentu adalah *null*, dianggap bahwa nilainya hilang, karena setiap dosen harus memiliki nama. Nilai *null* untuk atribut nomor rumah dapat berarti bahwa alamat tidak termasuk nomor rumah (tidak berlaku), bahwa nomor rumah ada tetapi tidak diketahui (hilang), atau bahwa tidak tahu apakah atau nomor rumah adalah bagian dari alamat dosen atau tidak (tidak diketahui)[7].

4.4. Relasi

Sebuah relasi adalah asosiasi antara beberapa entitas. Sebagai contoh, bisa didefinisikan relasi penasehat yang mengaitkan Dosen Supriadi dengan mahasiswa Amelia. Hubungan ini menentukan bahwa Supriadi adalah penasehat bagi mahasiswa Amelia. Sebuah set relasi adalah satu set hubungan dari jenis yang sama. Secara formal, ini adalah

[7] Adyanata Lubis, *Basis Data Dasar* (Deepublish, 2016).

ahubungan matematika pada set entitas $n \geq 2$. Jika E_1, E_2, ..., E_n adalah himpunan entitas, maka himpunan relasi R adalah himpunan bagian dari

$$\{(e_1, e_2, \ldots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$$

di mana (e_1, e_2, ..., e_n) adalah suatu relasi. Pertimbangkan dua set entitas dosen dan mahasiswa pada gambar di bawah ini. Didefinisikan set relasi penasehat untuk menunjukkan hubungan antara dosen dan mahasiswa. Gambar berikut menggambarkan hubungan ini.

100021	Supriadi	072119262	Amelia
200312	Sriani	072219034	Putri
100107	Bambang	082119921	Kirana
310201	Fadli	082119729	Febriana
		072119031	Andre

Gambar 4.1. Set Relasi Penasehat

Sebagai contoh lain, pertimbangkan entitas dua set mahasiswa dan kelas. Dapat didefinisikan hubungan yang diperlukan untuk menunjukkan hubungan antara seorang mahasiswa dan kelas di mana siswa tersebut terdaftar. Asosiasi antara set entitas disebut sebagai partisipasi, itu adalah entitas menetapkan E_1, E_2, ..., E_n berpartisipasi dalam relasi set R. Sebuah relasi dalam skema ER mewakili hubungan antara entitas yang disebutkan di perusahaan dunia nyata yang dimodelkan. Sebagai contoh, individu entitas

dosen Supriadi, yang memiliki ID dosen 100021, dan entitas mahasiswa Amelia, yang memiliki NIM 072119262, berpartisipasi dalam contoh relasi penasihat. Ini contoh relasi yang menyatakan bahwa di universitas, dosen Supriadi menasihati mahasiswa Amelia. Fungsi yang dimainkan suatu entitas dalam suatu hubungan disebut peran entitas.

Karena set entitas yang berpartisipasi dalam set hubungan umumnya berbeda, peran tersirat dan biasanya tidak ditentukan. Namun, mereka berguna saat arti suatu hubungan perlu klarifikasi. Seperti halnya ketika entitas menetapkan hubungan tidak berbeda, yaitu set entitas yang sama berpartisipasi dalam hubungan yang ditetapkan lebih dari sekali, dalam peran yang berbeda. Dalam jenis relasi ini, kadang-kadang disebut set relasi rekursif, nama peran eksplisit diperlukan untuk menentukan bagaimana suatu entitas berpartisipasi dalam relasi.

Sebagai contoh, pertimbangkan entitas mata kuliah yang mencatat informasi tentang semua mata kuliah yang ditawarkan di Universitas. Untuk menggambarkan situasi di mana satu mata kuliah (C2) merupakan prasyarat untuk mata kuliah lain (C1) dimiliki set relasi prasyarat yang dimodelkan dengan dipasangkan pada entitas. Mata kuliah pertama dari pasangan yang mengambil peran tentu saja C1, sedangkan yang kedua mengambil peran prasyarat C2. Dengan cara ini, semua relasi prasyarat ditandai

oleh pasangan (C1, C2), pasangan (C2, C1) tidak termasuk.

Dimungkinkan untuk memiliki lebih dari satu set relasi yang melibatkan set entitas yang sama. Sebagai contoh, set entitas dosen dan entitas mahasiswa berpartisipasi dalam set relasi penasihat. Selain itu, anggaplah setiap mahasiswa harus memiliki dosen yang lain yang melayani sebagai penasihat jurusan. Kemudian yang entitas dosen dan mahasiswa dapat berpartisipasi dalam set relasi lain, yaitu penasihat jurusan. Set relasi penasihat dan penasihat jurusan memberikan contoh set relasi biner, yaitu yang melibatkan dua set entitas. Sebagian besar set relasi dalam sistem basis data adalah biner. Namun, terkadang, set relasi melibatkan lebih dari dua set entitas.

Sebagai contoh, misalkan dimiliki set entitas proyek yang mewakili semua proyek penelitian yang dilakukan di universitas. Pertimbangkan set entitas dosen, mahasiswa dan proyek. Setiap proyek dapat memiliki beberapa mahasiswa dan beberapa dosen yang terkait. Selanjutnya, setiap mahasiswa mengerjakan proyek harus memiliki dosen terkait yang memandu mahasiswa dalam proyek tersebut. Untuk saat ini, diabaikan dua relasi pertama, antara proyek dan dosen, dan antara proyek dan mahasiswa. Sebaliknya, fokus pada informasi tentang dosen sedang membimbing mahasiswa yang mana pada proyek tertentu. Untuk mewakili informasi ini, dihubungkan tiga set entitas melalui set relasi pembimbing proyek, yang

menunjukkan bahwa mahasiswa tertentu dibimbing oleh dosen tertentu pada proyek tertentu.

Perhatikan bahwa seorang mahasiswa dapat memiliki dosen berbeda sebagai pembimbing untuk proyek yang berbeda, yang tidak dapat ditangkap oleh relasi biner antara mahasiswa dan dosen. Jumlah set entitas yang berpartisipasi dalam satu set relasi adalah derajat dari set relasi. Set relasi biner adalah derajat 2, set relasi ternary adalah derajat 3.

4.5. Kardinalitas

Memetakan kardinalitas, atau rasio kardinalitas, menyatakan jumlah entitas yang dapat dikaitkan dengan entitas lain melalui set relasi. Memetakan kardinalitas paling berguna dalam menggambarkan set relasi biner, meskipun mereka dapat berkontribusi pada deskripsi set relasi yang melibatkan lebih dari dua set entitas. Pada bagian ini, akan berkonsentrasi hanya pada set relasi biner. Untuk relasi biner, set R antara entitas set A dan B, pemetaan kardinalitas harus salah satu dari yang berikut:

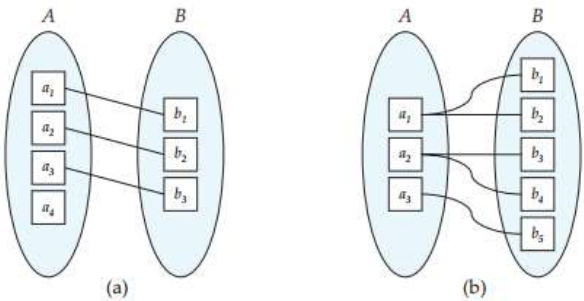
- Satu-ke-satu
  Suatu entitas di A dikaitkan dengan paling banyak satu entitas di B, dan sebuah entitas di B berhubungan dengan paling banyak satu entitas di A.
- Satu-ke-banyak
  Entitas dalam A dikaitkan dengan beberapa (nol atau lebih) entitas di B. Entitas dalam B,

hanya dapat dikaitkan dengan paling banyak satu entitas di A.

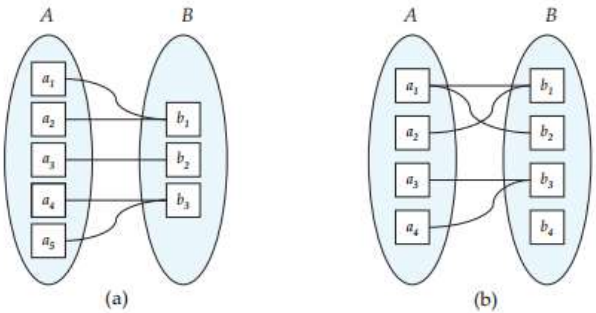
- Banyak-ke-satu
  Sebuah entitas di A berhubungan dengan paling banyak satu entitas di B. Sebuah entitas dalam B, dapat dikaitkan dengan beberapa (nol atau lebih) dari entitas di A.
- Banyak-ke-banyak
  Entitas dalam A dikaitkan dengan beberapa(nol atau lebih)entitas dalam B, dan entitas dalam B juga dapat dikaitkan dengan beberapa (nol atau lebih) entitas di A.

Kardinalitas merupakan pemetaan yang tepat untuk suatu relasi tertentu yang sangat jelas tergantung pada situasi dunia nyata yang ditetapkan oleh set relasi. Sebagai contoh, rangkaian relasi penasihat. Jika secara khusus pada universitas, seorang mahasiswa dapat dinasihati oleh hanya satu dosen, dan seorang dosen dapat membimbing beberapa mahasiswa, maka hubungan yang ditetapkan dari dosen ke mahasiswa adalah satu-ke-banyak. Jika seorang siswa dapat dinasihati oleh beberapa dosen (seperti dalam kasus mahasiswa dibimbing

bersama), maka set relasinya adalah banyak-ke-banyak.



Gambar 4.2. Pemetaan Kardinalitas (a)satu-ke-satu (b)satu-ke-banyak

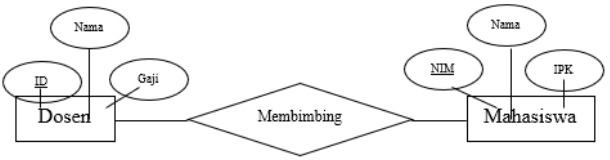


Gambar 4.3. Pemetaan Kardinalitas (a)banyak-ke-satu (b)banyak-ke-banyak

4.6. Entity Relationship Diagram (ERD)

ERD adalah kualitas yang sederhana dan jelas yang mungkin menjelaskan sebagian besar penggunaan model ER secara luas. ERD dapat mengekspresikan keseluruhan logika struktur database secara grafis.

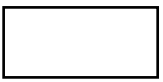
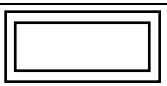
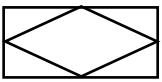
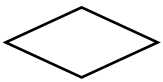
4.6.1. Struktur Dasar ERD

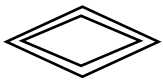
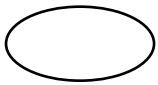
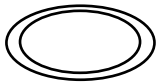
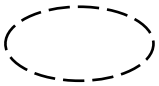


Gambar 4.4. ERD Dosen dan Mahasiswa

Pada gambar di atas dapat dilihat diagram relasi antara entitas dosen dan mahasiswa yang diberi nama relasi membimbing. Diagram ini menjelaskan hubungan dosen membimbing mahasiswa dan mahasiswa dibimbing oleh dosen. Bentuk persegi menggambarkan entitas, belah ketupat menggambarkan relasi, dan lingkaran menggambarkan atribut dari entitas. Untuk simbol ERD lebih lengkapnya dapat dilihat pada tabel di bawah ini.

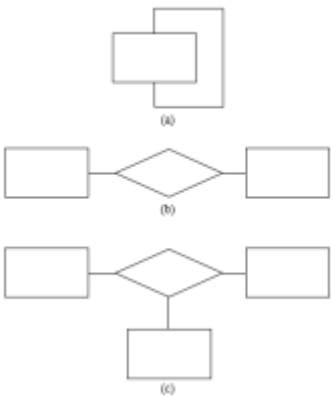
Tabel 4.3. Simbol Dasar ERD

Simbol	Keterangan
	Entitas Kuat
	Entitas Lemah
	Entitas Asosiatif
	Relasi

	Relasi Pengidentifikasi
	Atribut
	Atribut Bernilai Banyak
	Atribut Turunan

4.6.2. Derajat Relasi

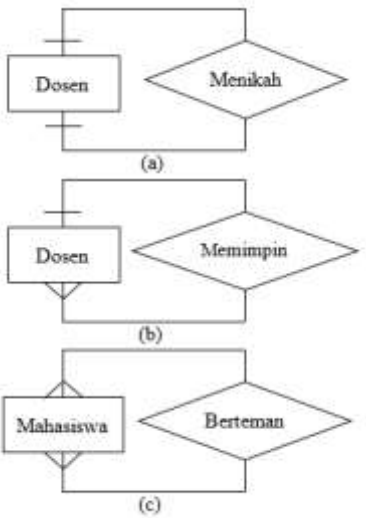
Derajat relasi adalah jumlah entitas yang berpartisipasi dalam suatu relasi. Sebagai contoh, dalam relasi mahasiswa mengambil mata kuliah, derajat relasinya adalah dua, karena ada dua entitas yang terlibat dalam relasi tersebut, yaitu entitas mahasiswa dan mata kuliah. Derajat relasi dalam ERD umumnya terbagi dalam tiga jenis sebagai berikut:



Gambar 4.5. Derajat Relasi (a)Unary (b)Binary (c)Ternary

a) Unary

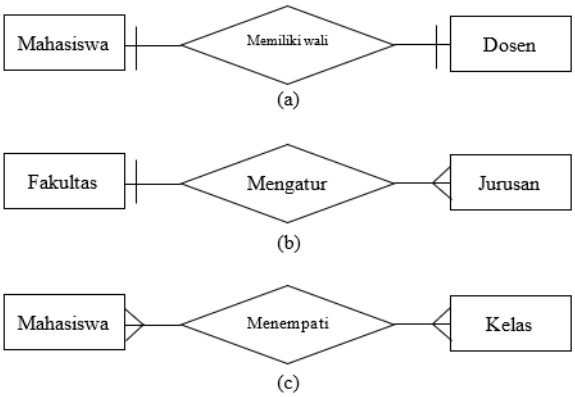
Relasi berderajat satu (*Unary relationship*) disebut juga relasi rekursif, yaitu relasi dimana entitas yang terlibat hanya satu. Sebagai contoh dosen menikah dengan dosen lainnya (relasi satu-ke-satu) yang berarti bahwa satu dosen hanya bisa menikah dengan satu dosen lainnya. Dosen memimpin dosen lainnya (relasi satu-ke-banyak) yang berarti bahwa satu dosen dapat memimpin banyak dosen lain. Mahasiswa berteman dengan mahasiswa lainnya (relasi banyak-ke-banyak) yang berarti banyak mahasiswa yang dapat berteman dengan banyak mahasiswa lainnya.



Gambar 4.6. *Unary* (a)satu-ke-satu (b)satu-ke-banyak (c)banyak-ke-banyak

b) Binary

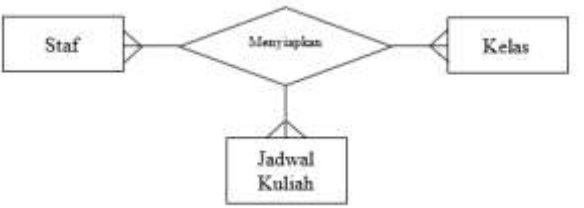
Relasi berderajat dua (*Binary Relationship*) yaitu relasi dimana entitas yang terlibat ada dua. Sebagai contoh relasi memiliki wali antara mahasiswa dengan dosen (relasi satu-ke-satu) yang berarti bahwa satu mahasiswa hanya bisa memiliki satu dosen wali. Fakultas mengatur jurusan (relasi satu-ke-banyak) yang berarti satu fakultas dapat mengatur banyak jurusan, namun satu jurusan hanya dapat diatur oleh satu fakultas. Mahasiswa menempati kelas (relasi banyak-ke-banyak) yang berarti bahwa satu mahasiswa dapat menempati banyak kelas dan satu kelas dapat ditempati oleh banyak mahasiswa.



Gambar 4.7. *Binary* (a)satu-ke-satu (b)satu-ke-banyak (c)banyak-ke-banyak

c) Ternary

Relasi berderajat tiga (*Ternany Relationship*) yaitu relasi dimana entitas yang terlibat ada tiga.



Gambar 4.8. *Ternary*

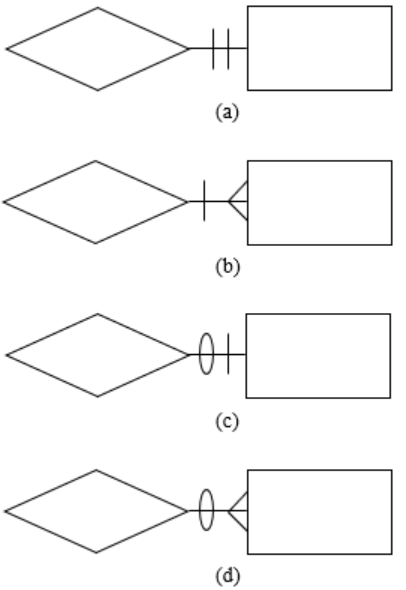
Pada gambar di atas, terlihat relasi berderajat tiga antara staf, jadwal kuliah dan kelas yang dihubungkan dengan relasi menyiapkan. Banyak staf dapat menyiapkan banyak jadwal kuliah ke banyak kelas.

4.6.3. Batasan Kardinalitas

Batasan kardinalitas adalah jumlah instansiasi yang dapat atau harus dilakukan oleh sebuah entitas terhadap entitas lainnya. Sebagai contoh, entitas dosen dengan mahasiswa dengan relasi menasehati. Syarat dosen menasehati mahasiswa dapat dibatasi misalnya satu dosen dapat menasehati banyak mahasiswa (relasi satu-ke-banyak). Namun, seorang dosen bisa saja tidak menasehati seorang mahasiswapun, karena itu diperlukan notasi yang lebih tepat untuk menggambarkan rentang kardinalitas untuk relasi tersebut.

Gambar 4.9. Relasi dengan batasan kardinalitas

a. Kardinalitas minimum
   Kardinalitas minimum adalah jumlah minimum sebuah entitas berasosiasi dengan entias lain. Seperti contoh pada gambar di atas, seorang dosen bisa saja tidak menjadi dosen penasehat, sehingga kardinalitas minimumnya dapat digambarkan dalam bentuk lingkaran kosong. Sementara itu, kardinalitas minimum pada sisi mahasiswa adalah satu, yang berarti bahwa satu mahasiswa wajib dinasehati oleh setidaknya satu orang dosen.
b. Kardinalitas maksimum
   Kardinalitas maksimum adalah jumlah maksimum sebuah entitas berasosiasi dengan entias lain. Seperti contoh pada gambar di atas, kardinalitas untuk entitas mahasiswa adalah banyak (lebih dari satu), yang menyatakan satu dosen mungkin menasehati lebih dari satu mahasiswa dan seorang mahasiswa hanya bisa memiliki satu dosen penasehat.



Gambar 4.10. Kardinalitas Relasi
(a)Mandatory One (b)Mandatory Many
(c)Optional One (d)Optional Many

4.7. Latihan

1. Apa yang dimaksud dengan entitas? Dan apa saja jenis-jenisnya?
2. Apa yang dimaksud dengan atribut? Dan apa saja jenis-jenisnya?
3. Apa yang dimaksud dengan relasi?
4. Gambarkan sebuah ERD yang menujukkan hubungan antara perusahaan dengan pelanggan.

BAB V

Perancangan Basis Data

Desain basis data yang baik sangat penting untuk memastikan bahwa data yang ingin disimpan konsisten dan dapat diambil atau diperbarui dengan cara yang paling efisien. Masalah desain basis data berkaitan dengan pemasangan sepotong data tertentu ke dalam organisasi logis yang sesuai. Beberapa faktor perlu dipertimbangkan selama proses, beberapa di antaranya adalah sebagai berikut.

- Berapa banyak tabel yang harus dimiliki?
- Bagaimana mereka harus saling terkait?
- Berapa banyak kolom yang harus dimiliki setiap tabel?
- Kolom mana yang harus menjadi kolom kunci?
- Kendala apa yang harus dimiliki?

Seperti sebelumnya, akan dipusatkan perhatian pada basis data relasional. Terlebih lagi, ketika berbicara tentang desain basis data, maka berbicara tentang aspek logis dari desain dan bukan aspek fisiknya.

5.1. Masalah Penyimpanan Data

Skema relasional menghadapi beberapa masalah yang tidak diinginkan.

1. Redundansi
   Tujuan dari sistem basis data adalah untuk mengurangi redundansi, yang berarti bahwa data hanya disimpan satu kali. Menyimpan

data/informasi berkali-kali menyebabkan pemborosan ruang penyimpanan dan peningkatan ukuran total data yang disimpan.

Tabel 5.1. Contoh Tabel Redundansi

Nama	Jurusan	Mata Kuliah	Dosen	Nilai
Ananda	Ilkomp	Sistem Basis Data	Raissa	A
Ananda	Ilkomp	Pemrograman	Armansyah	B
Ananda	Ilkomp	Pengolahan Citra	Sriani	A
Hendra	SI	Sistem Basis Data	Raissa	A
Hendra	SI	Pemrograman	Triase	B
Bambang	SI	Pemrograman	Triase	C

Pembaruan ke database dengan redundansi yang demikian berpotensi menjadi tidak konsisten. Pada tabel di atas nama dan jurusan seorang mahasiswa disimpan berkali-kali dalam basis data. Hal ini disebut redundansi data dalam basis data.

2. Anomali Pembaruan
   Beberapa salinan dari fakta yang sama dapat menyebabkan anomali pembaruan atau inkonsistensi. Saat pembaruan dilakukan dan hanya beberapa salinan ganda yang diperbarui. Jadi, perubahan pada Jurusan "Ananda" harus dibuat konsisten, dalam semua *record* yang berkaitan dengan mahasiswa "Ananda". Jika satu dari tiga *record* tidak berubah, maka akan ada ketidakkonsistenan dalam data.
3. Anomali Penyisipan

Jika tabel ini adalah satu-satunya relasi dalam basis data yang menunjukkan hubungan antara mahasiswa dan mata kuliah yang dia ajarkan, fakta bahwa seorang dosen yang diberikan mengajar dalam mata kuliah yang diberikan tidak dapat dimasukkan dalam basis data kecuali jika seorang mahasiswa terdaftar dalam mata kuliah.

4. Anomali Penghapusan
   Jika satu-satunya mahasiswa yang terdaftar dalam kursus yang diberikan menghentikan mata kuliah, informasi dosen akan hilang, jika ini adalah satu-satunya relasi dalam basis data yang menunjukkan hubungan antara mahasiswa dengan dosen yang mengajar.

5.2. Proses Perancangan Basis Data

Proses perancangan basis data dapat dibagi ke dalam langkah berikut:

1. Analisis Kebutuhan
   Langkah pertama dalam mendesain basis data aplikasi adalah memahami data apa yang akan disimpan dalam basis data, aplikasi apa yang harus dibangun di atasnya, dan operasi apa yang paling sering dan tunduk pada persyaratan kinerja. Dengan kata lain, harus dicari tahu apa yang diinginkan pengguna dari basis data. Ini biasanya merupakan proses informal yang melibatkan diskusi dengan kelompok pengguna, studi tentang lingkungan operasi saat ini dan bagaimana perubahan yang diharapkan, analisis dokumentasi yang

tersedia tentang aplikasi yang ada yang diharapkan akan diganti atau dilengkapi dengan basis data, dan sebagainya. Beberapa metodologi telah diusulkan untuk mengatur dan menyajikan informasi yang dikumpulkan dalam langkah ini, dan beberapa alat otomatis telah dikembangkan untuk mendukung proses ini.

2. Perancangan Basis Data Konseptual
   Informasi yang dikumpulkan dalam langkah analisis persyaratan digunakan untuk mengembangkan deskripsi tingkat tinggi dari data yang akan disimpan dalam basis data, bersama dengan kendala yang diketahui menghambat data ini. Langkah ini sering dilakukan dengan menggunakan model ER. Model ER adalah salah satu dari beberapa model data tingkat tinggi, atau semantik, yang digunakan dalam desain basis data. Tujuannya adalah untuk membuat deskripsi sederhana dari data yang sangat cocok dengan cara pengguna dan pengembang memikirkan data (dan orang-orang dan proses yang akan diwakili dalam data). Ini memfasilitasi diskusi di antara semua orang yang terlibat dalam proses desain, bahkan mereka yang tidak memiliki latar belakang teknis. Pada saat yang sama, desain awal harus cukup tepat untuk memungkinkan terjemahan langsung ke dalam model data yang didukung oleh database komersial sistem.
3. Desain Basis Data Logis

Harus dipilih DBMS untuk mengimplementasikan desain basis data, dan mengubah desain basis data konseptual menjadi skema basis data dalam model data DBMS yang dipilih. Hanya akan dipertimbangkan DBMS relasional, dan oleh karena itu, tugas dalam langkah desain logis adalah untuk mengubah skema ER menjadi skema database relasional[8].

5.3. Latihan

1. Jelaskan langkah-langkah yang perlu dilakukan dalam merancang sebuah basis data.
2. Jelaskan bagaimana basis data dapat mengurangi redudansi data.
3. Jelaskan bagaimana basis data dapat mengatasi berbagai anomali dalam penyimpanan data.

[8] Vijay Krishna Pallaw, *Database Management Systems Second Edition* (Asian Books Private Limited, 2010).

BAB VI

Normalisasi

Normalisasi adalah proses dimana dapat mendekomposisi atau membagi relasi menjadi lebih dari satu relasi untuk menghilangkan anomali dalam database relasional. Ini adalah proses langkah demi langkah dan setiap langkah dikenal sebagai bentuk normal.

Istilah penting dalam teknik normalisasi:

- *Field* / atribut kunci
  Setiap *file* selalu terdapat kunci dari *file* berupa satu *field* atau satu *field* yang dapat mewakili *record*
- *Candidate key*
  Kumpulan atribut minimal yang secara unik mengidentifikasi sebuah baris yang fungsinya sebagai calon *primary key*.
- *Composite key*
  Kunci kandidat yang berisi lebih dari satu atribut
- *Primary key*
  Candidate key yang dipilih untuk mengidentifikasi baris secara unik
- *Alternate key*
  Candidate key yang tidak dipilih sebagai *primary key*
- *Foreign key*
  Kunci di tabel lain yang terhubung dengan *primary key* pada sebuah tabel

6.1. Manfaat Normalisasi

Manfaat normalisasi yaitu:

1. Menghasilkan tabel yang lebih kecil dengan baris yang lebih kecil
2. Pencarian, pengurutan, dan membuat indeks lebih cepat, karena tabel lebih sempit, dan lebih banyak baris cocok pada halaman data.
3. Menghasilkan lebih banyak tabel dengan memecah tabel asli. Dengan demikian bisa ada lebih banyak indeks berkerumun dan karenanya ada lebih banyak fleksibilitas dalam menyetel kueri.
4. Pencarian indeks umumnya lebih cepat karena indeks cenderung lebih sempit dan lebih pendek.
5. Semakin banyak tabel memungkinkan penggunaan segmen yang lebih baik untuk mengontrol penempatan fisik data.
6. Ada lebih sedikit indeks per tabel dan karenanya perintah modifikasi data lebih cepat.
7. Ada sejumlah kecil nilai nol dan redundansi. Ini membuat basis data lebih kompak.
8. Anomali modifikasi data berkurang.
9. Secara konseptual lebih bersih dan lebih mudah untuk dipertahankan dan diubah seiring perubahan kebutuhan.

6.2. Ketergantungan dalam Normalisasi

Ketergantungan (*dependency*) merupakan konsep yang mendasari normalisasi. Dalam basis

data *dependency* lebih sering disebut *Functional Dependency* atau Ketergantungan Fungsional yang digunakan untuk menggambarkan hubungan, batasan, keterkaitan antara atribut-atribut dalam relasi. Atau lebih jelasnya nilai dari suatu atribut dapat menentukan nilai dari atribut yang lain. *Dependency* akan mencari acuan untuk pendekomposisian data ke dalam bentuk yang paling efisien. Sebagai contoh yaitu untuk NIM dan Nama_Mhs. NIM secara fungsional menentukan Nama_Mhs, karena untuk setiap NIM yang sama maka nilai Nama_Mhs nya sama. *Dependency* dapat dibagi ke dalam tiga jenis, yaitu:

1. *Full Dependency*
   Menunjukan jika terdapat atribut A dan B dalam suatu relasi, dimana B memiliki ketergantungan fungsional secara penuh pada A, tapi B tidak memiliki ketegantungan terhadap subset A.
2. *Partial Dependency*
   Ketergantungan parsial atau sebagian memiliki 2 atribut dari A untuk menentukan B, namun untuk menentukan B tidak harus 2 atribut artinya jika salah satu atribut A yang menentukan B dapat dihilangkan namun tidak merubah arti relasi dan masih tetap berelasi ketergantungan.
3. *Transitive Dependency*
   Transitive dependency biasanya terjadi pada tabel hasil relasi, atau kondisi dimana terdapat tiga atribut A,B,C. Kondisinya adalah A tergantung terhadap

B dan B tergantung terhadap C. Maka C dikatakan sebagai *transitive dependency* terhadap A melalui B.

6.3. Tahapan Normalisasi

1. *Unnormalized*
   Pada tahap ini, diambil seluruh data yang ada dan diperlukan dalam database itu sendiri. Misalnya pada contoh tabel nilai mahasiswa, datanya terdiri dari NIM, nama mahasiswa, alamat mahasiswa, ID Dosen, nama dosen, status dosen, nilai basis data, nilai pemrograman, nilai pengantar ilmu komputer, nilai interaksi manusia dan komputer.

Tabel 6.1. Tabel Mentah

NIM	Nama_Mhs	Alamat_Mhs	ID_Dosen	Nama_Dosen	Status_Dosen	Basis Data	Pemrograman	PIK	IMK
072134321	Andi	Medan	10010	Raissa	Tetap	78		92	
072120089	Budi	Medan	10087	Andini	Honorer		85	66	78
072129262	Iwan	Binjai	10091	Ulfayani	Tetap	56	83		
087201231	Sari	Medan	10091	Ulfayani	Tetap		96		83
082324321	Putri	Medan	10003	Laylan	Honorer	89			77
079909032	Melisa	Tj.Morawa	10003	Laylan	Honorer	50		41	

Tabel mentah di atas memerlukan suatu analisa dikarenakan tabel yang ada di soal mempunyai suatu keanehan. Perhatikan pada kolom Basis Data, Pemrograman, PIK, dan IMK. Setiap nama kolom yang sudah disebutkan memiliki kode tersendiri yang bisa merepresentasikan nama dari mata kuliah. Dengan begitu bisa dikumpulkan kode mata kuliah menjadi satu kolom baru yang bisa diberi nama Kode_Matkul. Kemudian nama dari mata kuliah juga bisa dibuat menjadi satu

kolom baru yang diberi nama Nama_Matkul sehingga bentuk tabel seperti yang terlihat pada tabel di bawah ini.

Tabel 6.2. Tabel *Unnormalized*

NIM	Nama_Mhs	Alamat_Mhs	ID_Dosen	Nama_Dosen	Status_Dosen	Kode_Matkul	Nama_Matkul	Nilai
072134321	Andi	Medan	10010	Raissa	Tetap	MK001	Basis Data	78
						MK003	PIK	92
072120089	Budi	Medan	10087	Andini	Honorer	MK002	Pemrograman	85
						MK003	PIK	66
						MK004	IMK	78
072129262	Iwan	Binjai	10091	Ulfayani	Tetap	MK001	Basis Data	56
						MK002	Pemrograman	83
087201231	Sari	Medan	10091	Ulfayani	Tetap	MK002	Pemrograman	96
						MK004	IMK	83
082324321	Putri	Medan	10003	Laylan	Honorer	MK001	Basis Data	89
						MK004	IMK	77
079909032	Melisa	Tj.Morawa	10003	Laylan	Honorer	MK001	Basis Data	50
						MK003	PIK	41

2. Normal Satu (1NF)
   Pada tahap ini, bagi seluruh data yang diperlukan menjadi beberapa bagian berdasarkan jenis data tersebut. Sebuah tabel bisa dikategorikan sebagai tabel 1NF jika di setiap baris *record* hanya memiliki 1 *value*. Hasil transformasi dari *Unnormalized Form* ke 1NF bisa dilihat pada tabel di bawah ini:

Tabel 6.3. Tabel Normal 1 (1NF)

NIM	Nama_Mhs	Alamat_Mhs	ID_Dosen	Nama_Dosen	Status_Dosen	Kode_Matkul	Nama_Matkul	Nilai
072134321	Andi	Medan	10010	Raissa	Tetap	MK001	Basis Data	78
072134321	Andi	Medan	10010	Raissa	Tetap	MK003	PIK	92
072120089	Budi	Medan	10087	Andini	Honorer	MK002	Pemrograman	85
072120089	Budi	Medan	10087	Andini	Honorer	MK003	PIK	66
072120089	Budi	Medan	10087	Andini	Honorer	MK004	IMK	78
072129262	Iwan	Binjai	10091	Ulfayani	Tetap	MK001	Basis Data	56
072129262	Iwan	Binjai	10091	Ulfayani	Tetap	MK002	Pemrograman	83
087201231	Sari	Medan	10091	Ulfayani	Tetap	MK002	Pemrograman	96
087201231	Sari	Medan	10091	Ulfayani	Tetap	MK004	IMK	83
082324321	Putri	Medan	10003	Laylan	Honorer	MK001	Basis Data	89
082324321	Putri	Medan	10003	Laylan	Honorer	MK004	IMK	77
079909032	Melisa	Tj.Morawa	10003	Laylan	Honorer	MK001	Basis Data	50
079909032	Melisa	Tj.Morawa	10003	Laylan	Honorer	MK003	PIK	41

3. Normal Dua (2NF)
   Pada tahap ini, bagi berdasarkan jenis dan memberikan primary key pada masing-masing tabel. Sebuah tabel bisa dikategorikan sebagai tabel 2NF jika tabel sudah dalam keadaan 1NF dan setiap kolom didalam tabel itu *functional dependency* kepada semua *key* (*Full Dependency*), bukan hanya kepada salah satu *key* (*Partial Dependency*). Berarti harus ditentukan terlebih dahulu *key* yang ada di dalam tabel. Dari tabel di atas bisa ditentukan bahwa ada dua *candidate key* yang tersedia dalam tabel diatas, yaitu NIM + Kode_Matkul, dan NIM + Nama_Matkul. Setelah dipilih dari kedua *candidate key*, maka yang paling efisien untuk menjadi *Primary Key* adalah NIM + Kode_Matkul.

   Setelah itu, tentukan *Functional Dependency* yang ada di dalam tabel yaitu sebagai berikut:

Tabel 6.4. Functional Dependency

Full Dependency	NIM + Kode_Matkul	Nama_Matkul Nilai_Matkul
Partial Dependency	NIM	Nama_Mhs Alamat_Mhs ID_Dosen Nama_Dosen Status_Dosen
Partial Dependency	Kode_Matkul	Nama_Matkul
Transitive Dependency	ID_Dosen	Nama_Dosen Status_Dosen

Setelah mendapatkan semua *Functional Dependency* di dalam tabel, maka untuk mengubah tabel 1NF menjadi 2NF harus dipindahkan kolom yang bergantung hanya kepada salah satu dari key (*Partial Dependency*) ke dalam tabel baru sehingga saat ini tabel terpisah seperti terlihat pada tabel di bawah ini.

Tabel 6.5. Tabel Mahasiswa 2NF

NIM	Nama_Mhs	Alamat_Mhs	ID_Dosen	Nama_Dosen	Status_Dosen
072134321	Andi	Medan	10010	Raissa	Tetap
072120089	Budi	Medan	10087	Andini	Honorer
072129262	Iwan	Binjai	10091	Ulfayani	Tetap
087201231	Sari	Medan	10091	Ulfayani	Tetap
082324321	Putri	Medan	10003	Laylan	Honorer
079909032	Melisa	Tj.Morawa	10003	Laylan	Honorer

Tabel 6.6. Tabel Mata Kuliah 2NF

Kode_Matkul	Nama_Matkul
MK001	Basis Data
MK002	Pemrograman
MK003	PIK
MK004	IMK

Tabel 6.7. Tabel Nilai 2NF

NIM	Kode_Matkul	Nilai
072134321	MK001	78
072134321	MK003	92
072120089	MK002	85
072120089	MK003	66
072120089	MK004	78
072129262	MK001	56
072129262	MK002	83
087201231	MK002	96
087201231	MK004	83
082324321	MK001	89
082324321	MK004	77
079909032	MK001	50
079909032	MK003	41

4. Normal Tiga (3NF)
   Pada tahap ini, bagi menjadi lebih terperinci untuk menghindari terjadinya redundansi. Sebuah tabel bisa dikategorikan sebagai tabel 3NF jika tabel sudah dalam 2NF dan setiap kolom yang bukan *key* harus *functional dependency* dengan *primary key* nya. Bisa dilihat didalam Tabel Nilai dan Tabel Mata

Kuliah sudah memenuhi persyaratan 3NF. Tetapi didalam Tabel Mahasiswa terdapat kolom yang bergantung bukan kepada *key* nya (*Transitive Dependency*) yaitu Nama_Dosen dan Status_Dosen. Ini menunjukan bahwa Tabel Mahasiswa belum memenuhi persyaratan menjadi 3NF. Nama_Dosen dan Status_Dosen itu *functional dependency* kepada ID_Dosen yang diketahui bahwa ID_Dosen itu bukanlah sebuah *primary key*. Maka untuk membuat Tabel Nilai kedalam bentuk 3NF, harus ditempatkan kolom yang *functional dependency* bukan kepada *key* nya (*Transitive Dependency*) kedalam tabel yang baru sehingga bentuk tabel menjadi seperti tabel di bawah ini:

Tabel 6.8. Tabel Mahasiswa 3NF

NIM	Nama_Mhs	Alamat_Mhs
072134321	Andi	Medan
072120089	Budi	Medan
072129262	Iwan	Binjai
087201231	Sari	Medan
082324321	Putri	Medan
079909032	Melisa	Tj.Morawa

Tabel 6.9. Tabel Dosen 3NF

ID_Dosen	Nama_Dosen	Status_Dosen
10010	Raissa	Tetap
10087	Andini	Honorer
10091	Ulfayani	Tetap
10003	Laylan	Honorer

Tabel 6.10. Tabel Mata Kuliah 3NF

Kode_Matkul	Nama_Matkul
MK001	Basis Data
MK002	Pemrograman
MK003	PIK
MK004	IMK

Tabel 6.11. Tabel Nilai 3NF

NIM	Kode_Matkul	Nilai
072134321	MK001	78
072134321	MK003	92
072120089	MK002	85
072120089	MK003	66
072120089	MK004	78
072129262	MK001	56
072129262	MK002	83
087201231	MK002	96
087201231	MK004	83
082324321	MK001	89
082324321	MK004	77
079909032	MK001	50
079909032	MK003	41

6.4. Latihan

Carilah sebuah bon faktur manual, lakukan normalisasi terhadap tabel pada bon faktur tersebut.

BAB VII

Query

Pada bab sebelumnya telah dipelajari bagaimana normalisasi akhirnya menghasilkan tabel normal ketiga yang dapat diimplementasikan pada basis data. Langkah berikutnya ketikkan kueri pada basis data untuk menambahkan tabel – tabel yang telah normal tersebut. Pada contoh berikut ini digunakan MySQL sebagai *Database Management System* (DBMS).

7.1. Membuat Basis Data

```
create database universitas
```

7.2. Membuat Tabel

```
create table mahasiswa
(NIM varchar (10),
Nama_Mhs varchar (50) not
null,
Alamat_Mhs varchar (100),
primary key (NIM);


create table dosen
(ID_Dosen varchar (10),
Nama_Dosen varchar (50) not
null,
Status_Dosen varchar (10),
primary key (ID_Dosen);
```

```
    create table mata_kuliah
    (Kode_Matkul varchar (10),
    Nama_Matkul varchar (50) not
null,
    primary key (Kode_Matkul);


    create table nilai
    (NIM varchar (5),
    Kode_Matkul varchar (20) not
null,
    Nilai_akhir numeric (8,2),
    primary key (NIM),
    foreign   key   (Kode_Matkul)
    references mata_kuliah);
```

7.3. Menambah Data

```
    insert into mahasiswa (NIM,
Nama_Mhs, Alamat_Mhs)
    values ('072134321', 'Andi',
'Medan');

    insert into mahasiswa (NIM,
Nama_Mhs, Alamat_Mhs)
    values ('072120089', 'Budi',
'Medan');

    insert   into   nilai   (NIM,
Kode_Matkul, Nilai_Akhir)
    values     ('     072134321',
'MK001', 78);

    insert   into   nilai   (NIM,
Kode_Matkul, Nilai_Akhir)
```

```
    values     (’     072134321’,
’MK003’, 92);
```

7.4. Mengubah Data

```
    update nilai
    set Nilai_Akhir = 83
    where  NIM =  ’072120089’  and
Kode_Matkul = ’MK001’;

    update dosen
    set nama= ‘Laylan S’
    where ID_Dosen= ‘10003’;
```

7.5. Menghapus Data

```
    delete from dosen
    where ID_Dosen= ’10010’;
```

7.6. Latihan

1. Tuliskan kueri SQL untuk membuat basis data perpustakaan.
2. Tuliskan kueri SQL untuk menambahkan tabel buku yang terdiri dari kode, judul, pengarang, penerbit, isbn, dan kategori.
3. Tuliskan kueri untuk menambahkan daftar buku baru ke dalam tabel buku.

DAFTAR PUSTAKA

Date, Christopher J. “An Introduction to Database Systems 8th Edition,” 2004.

Garcia-Molina, Hector, Jeffrey D. Ullman, dan Jennifer Widom. *DATABASE SYSTEMS The Complete Book*. *Pearson Prentice Hall*. Vol. 26. New Jersey: PEARSON, 2009. http://www.worldcat.org/isbn/813170842X.

Gupta, Satinder Bal, dan Aditya Mittal. *Database Management System*. UNIVERSITY SCIENCE PRESS, 2017.

Korth, Henry F, dan Abraham Silberschatz. *Database System Concepts*. *Communications of the ACM*. Vol. 40, 1997. http://portal.acm.org/citation.cfm?doid=253671.253760.

Lubis, Adyanata. *Basis Data Dasar*. Deepublish, 2016.

Nugroho, Adi. *Perancangan dan Implementasi Sistem Basis Data*. Penerbit Andi, 2011.

Pallaw, Vijay Krishna. *Database Management Systems Second Edition*. Asian Books Private Limited, 2010.

Ramakrishnan, Raghu, dan Johannes Gehkre. *Database Management Systems Third Edition*. Mc Graw Hill, 2003.

Mata kuliah Basis Data merupakan mata kuliah yang memberikan konsep dasar basis data kepada mahasiswa, bagaimana membuat model data, merancang basis data dengan baik, normalisasi basis data serta menjelaskan sistem informasi dimana basis data dapat diterapkan. Tujuan dari mata kuliah ini adalah mahasiswa dapat merancang, membuat, dan mengelola basis data, baik yang berbasis komputer tunggal atau basis data berbasis web, serta mampu mengaplikasikannya dalam dunia kerja. Capaian Pembelajaran pada mata kuliah Basis Data yaitu diharapkan Mahasiswa mampu menjelaskan konsep dasar basis data, mampu merancang, membuat dan mengelola basis data dan mampu mengaplikasikan basis data ke dalam sistem. Adapun materi pembelajaran yang terdapat dalam buku ini yaitu Pengantar Basis Data, Basis Data Relasional, Structured Query Language (SQL), Entity Relationship (ER), Perancangan Basis Data, Normalisasi, dan Query.

Tentang Penulis

Raissa Amanda Putri, S.Kom.,M.TI.

Lahir di Kota Binjai, tanggal 10 Juli 1989. Telah menyelesaikan studi S1 jurusan Sistem Informasi di STMIK Mikroskil Medan pada tahun 2011 serta Magister Teknik Informatika di Universitas Bina Nusantara Jakarta pada tahun 2015. Mulai mengajar sejak tahun 2011 di STMIK Mikroskil dan mulai tahun 2018 hingga saat ini sebagai dosen tetap di Universitas Islam Negeri Sumatera Utara Medan.

Al Azhar Centre
Ikatan Alumni Universitas Al-Azhar Mesir
Jl. Jalan Jangka, Gang Roda, No.69, Medan
Telp : 082362218683
Email: azharcentre.publishing@gmail.com

ISBN 978-623-91257-7-6