Catatan Materi UTS
Basis Data (IF2240)

Semester II 2024/2025

12 April 2025

IF '23

Razi Rachman Widyadhana - *@zirachw*

Apabila terdapat kesalahan pada file, silakan kontak penulis :)

Dilarang membawa hardcopy saat ujian berlangsung

This page intentionally left blank

Daftar Isi

Bab 1

Introduction

1.1 Konsep Dasar

Definisi 1.1: Basis Data

Kumpulan data yang terorganisir dan dapat dikelola dengan sistem tertentu untuk mendukung pengambilan keputusan dan operasi bisnis.

Definisi 1.2: Data

Representasi tersimpan dari objek dan peristiwa yang bermakna.

Data dapat diklasifikasi menjadi:

1. **Structured Data:**

   Umumnya data tabular yang diwakili oleh kolom dan baris dalam database.

2. **Unstructured Data:**

   Data yang tidak diatur dengan cara yang sudah ditentukan sebelumnya atau tidak memiliki model data yang sudah ditentukan sebelumnya. Contohnya: video, audio, *image*, dan *binary* data *files*

3. **Semi-Structured Data:**

   Data yang tidak terdiri dari data terstruktur (basis data relasional) tetapi masih memiliki struktur. Contohnya: JavaScript Object Notation (JSON) format.

Definisi 1.3: Informasi

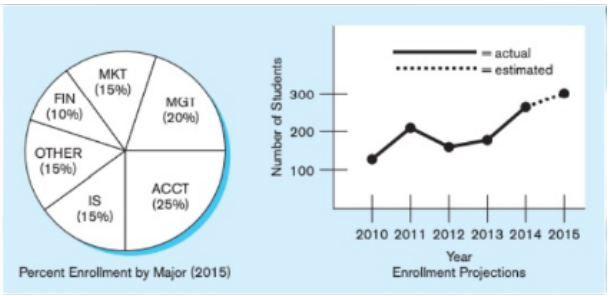
Data yang telah diproses dan diinterpretasikan sehingga memiliki makna bagi penggunanya.

Class Roster

Course: MGT 500 Business Policy Semester: Spring 2015

Section: 2

Name	ID	Major	GPA
Baker, Kenneth D.	324917628	MGT	2.9
Doyle, Joan E.	476193248	MKT	3.4
Finkle, Clive R.	548429344	PRM	2.8
Lewis, John C.	551742186	MGT	3.7
McFerran, Debra R.	409723145	IS	2.9
Sisneros, Michael	392416582	ACCT	3.3



Gambar 1.1: (a) Data in Context - Context membantu pengguna memahami data, **(b) Summarized Data** - Tampilan grafis yang mengubah data menjadi informasi yang berguna

Definisi 1.4: Metadata

Data yang mendeskripsikan karakteristik data lain, termasuk tipe data, struktur, dan hubungan antardata.

TABLE 1-1 Example Metadata for Class Roster

Data Item	Metadata					
Name	**Type**	**Length**	**Min**	**Max**	**Description**	**Source**
Course	Alphanumeric	30			Course ID and name	Academic Unit
Section	Integer	1	1	9	Section number	Registrar
Semester	Alphanumeric	10			Semester and year	Registrar
Name	Alphanumeric	30			Student name	Student IS
ID	Integer	9			Student ID (SSN)	Student IS
Major	Alphanumeric	4			Student major	Student IS
GPA	Decimal	3	0.0	4.0	Student grade point average	Academic Unit

Gambar 1.2: Example of Metadata

1.2 File Processing System vs The Database Approach

Ketika pemrosesan data berbasis komputer pertama kali tersedia, masih belum ada basis data. Agar dapat digunakan untuk aplikasi bisnis, komputer harus menyimpan, memanipulasi, dan mengambil file data yang besar.

Kelemahan pendekatan ini akibat **Data Dependency**:

1. Setiap pemrogram aplikasi harus memelihara data mereka sendiri.
2. Setiap program aplikasi perlu menyertakan kode untuk metadata setiap file.
3. Setiap program aplikasi harus memiliki rutinitas pemrosesan sendiri untuk *read*, *create*, *update* dan *delete* data
4. Kurangnya koordinasi dan kontrol pusat.

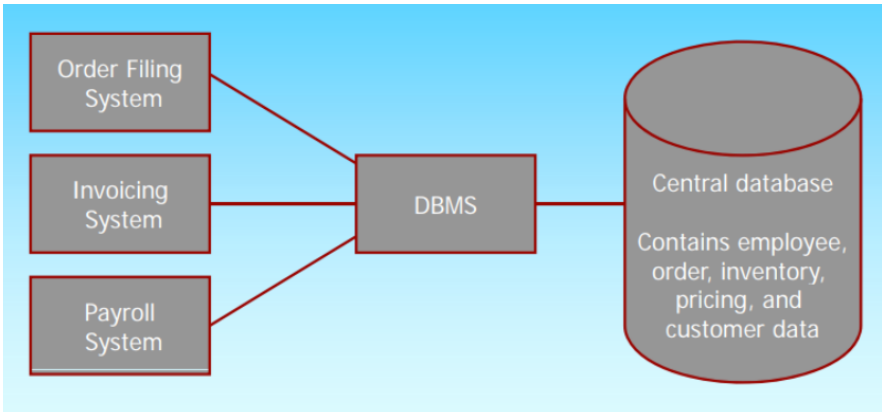
Kelemahan pendekatan ini akibat **Data Redundancy**:

1. Pemborosan ruang ruang karena sistem/program yang berbeda memiliki salinan terpisah dari data yang sama.
2. Ketika data berubah dalam satu file, dapat menyebabkan inkonsistensi
3. Integrity Constraint (misalnya, saldo akun > 0) menjadi "terkubur" dalam kode program daripada dinyatakan secara eksplisit. Kemudian, sulit untuk menambahkan batasan baru atau mengubah yang sudah ada

Sebagai solusinya, **pendekatan basis data** diperkenalkan dengan menggunakan **Database Management System (DBMS)** yang memberikan pengelolaan data yang lebih efisien, terstruktur, dan aman.

Definisi 1.5: Database Management System (DBMS)

Sistem perangkat lunak yang digunakan untuk membuat, memelihara, dan menyediakan akses terkontrol ke basis data pengguna.
DBMS mengelola sumber daya data seperti sistem operasi mengelola sumber daya perangkat keras.



Gambar 1.3: Example of DBMS

Kelebihan pendekatan Basis Data:

1. **Improved Data Sharing**
   Pengguna yang berbeda mendapatkan tampilan data yang berbeda.
2. **Enforcement of Standards**
   Pengguna yang berbeda mendapatkan tampilan data yang berbeda.
3. **Improved Data Quality**
   Constraints, data validation rules.
4. **Better Data Accessibility/Responsiveness**
   Penggunaan standard data query language (SQL).
5. **Security, Backup/Recovery, Concurrency**
   Pemulihan bencana lebih mudah.

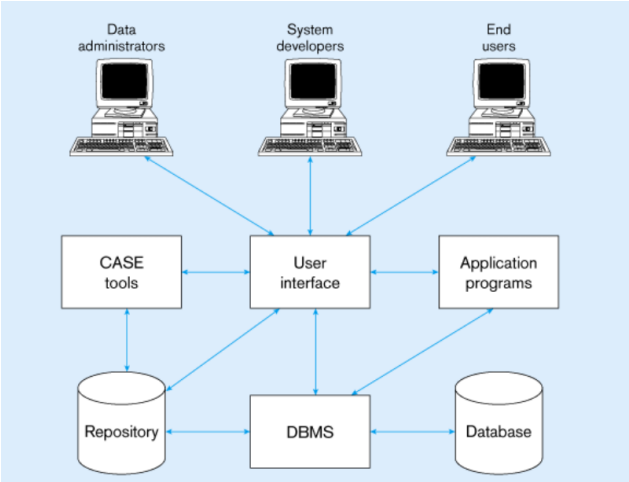
Costs dan Risks dari pendekatan Basis Data:

1. **Up-front costs**
   Biaya dan Kerumitan Manajemen Instalasi serta biaya konversi jika dalam tahap transisi.
2. **Ongoing Costs**
   Membutuhkan personel baru yang terspesialisasi dan membutuhkan pencadangan dan pemulihan eksplisit.
3. **Organizational Conflict**
   Kebiasaan lama yang sulit dihilangkan.

1.3 Pengaplikasian Basis Data

Lingkungan basis data terdiri dari beberapa komponen yang bekerja bersama untuk mengelola dan memanfaatkan data secara efisien. **Komponen utama dalam lingkungan basis data**:

1. **CASE Tools** - Computer-aided software engineering
2. **Repository** - centralized storehouse of metadata
3. **Database Management System (DBMS)** - software for managing the database
4. **Database** - storehouse of the data
5. **Application Programs** - software using the data
6. **User Interface** - text and graphical displays
7. **Data Administrators** - personnel responsible for maintaining the database to users
8. **System Developers** - personnel responsible for designing databases and software
9. **End Users** - personnel people who use the applications and databases



Gambar 1.4: Komponen di dalam lingkungan DBMS

Aplikasi dari Basis Data memiliki rentang-rentang berikut:

1. Personal databases
2. Workgroup databases
3. Departmental/ divisional databases
4. Enterprise database - Enterprise resource planning (ERP) systems dan implementasi *data warehousing*.

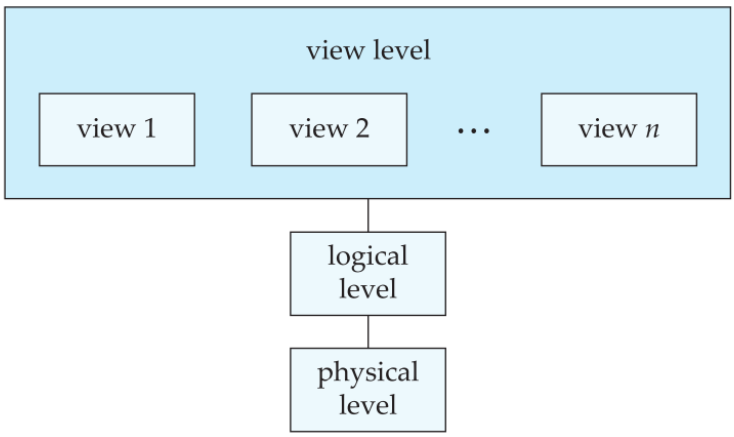
1.4 Level-Level Abstraksi

Konsep

- **Physical level**: Menjelaskan bagaimana sebuah record disimpan.
- **Logical level**: Menjelaskan data yang disimpan dalam database, dan hubungan di antara data tersebut.

```
type instructor = record
  ID : string;
  name :  string;
  dept_name :  string;
  salary :  integer;
end;
```

- **View level**: Program aplikasi menyembunyikan detail tipe data. View juga dapat menyembunyikan informasi (seperti gaji karyawan) untuk tujuan keamanan.



Gambar 1.5: Arsitektur untuk sistem basis data (Arsitektur ANSI/SPARC)

1.5 Instances dan Schemas

Definisi 1.6

Instance adalah konten aktual dari basis data pada titik waktu tertentu.

Schema adalah struktur dari sebuah Basis Data. Terdiri dari:

- **Logical Schema**: keseluruhan struktur logis dari basis data.
- **Physical Schema**: keseluruhan struktur fisik dari basis data.

1.6 Physical Data Independence

Konsep

Kemampuan untuk memodifikasi physical schema tanpa mengubah logical schema.

- Aplikasi bergantung pada logical schema
- Secara umum, antarmuka antara berbagai level dan komponen harus didefinisikan dengan baik sehingga perubahan di beberapa bagian tidak terlalu mempengaruhi bagian lainnya.

1.7 Data Definition Language

Definisi 1.7

Notasi untuk mendefinisikan skema database.
DDL Compiler menghasilkan satu set templat tabel disimpan dalam **kamus data** - berisi metadata (misalnya data tentang data)

- Skema basis data
- Integrity constraints: Primary key (ID yang secara unik mengidentifikasi instruktur)
- Authorization

1.8 Data Manipulation Language

Definisi 1.8

Bahasa untuk mengakses dan memperbarui data yang diorganisir oleh model data yang sesuai. Pada dasarnya ada dua jenis bahasa manipulasi data:

- **DML prosedural**

  Mengharuskan pengguna untuk menentukan data apa yang dibutuhkan dan bagaimana cara mendapatkan data tersebut.

- DML deklaratif - mengharuskan pengguna untuk menentukan data apa yang dibutuhkan tanpa menentukan bagaimana cara mendapatkan data tersebut.

1.9 Database Design

Konsep

Logical Design: Memutuskan skema basis data. Desain database mengharuskan kita untuk menemukan kumpulan skema relasi yang "baik".

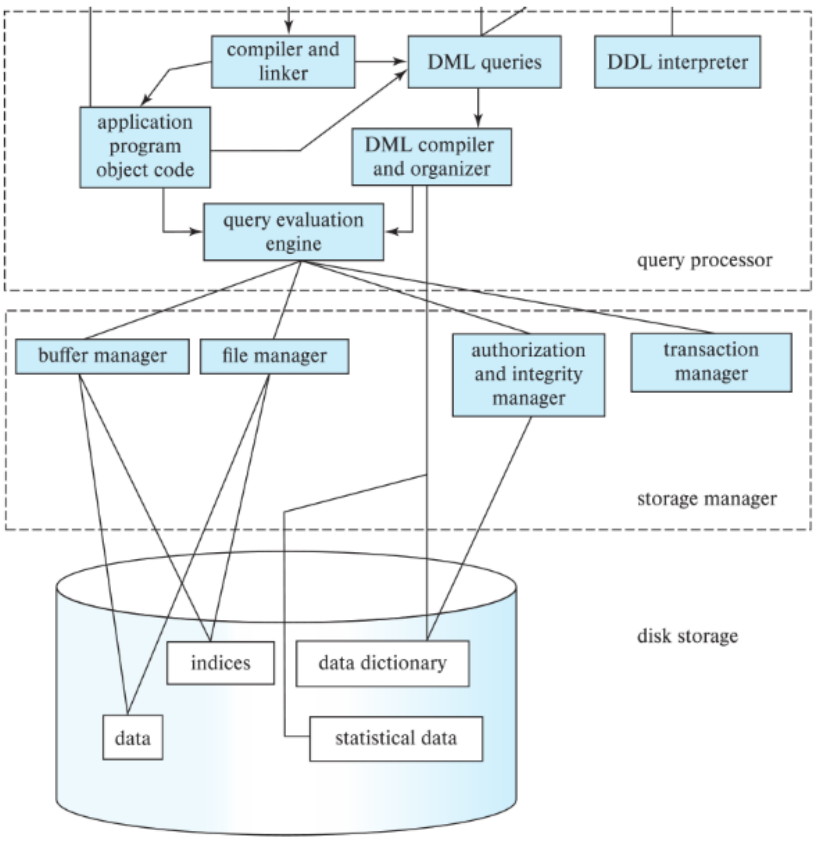
- **Business decision** – What attributes should we record?
- **Computer Science decision** – What attributes should we record?

Physical Design: Memutuskan tata letak fisik basis data

1.10 Database Management System

Sistem manajemen basis data dipartisi ke dalam modul-modul yang menangani masing-masing tanggung jawab dari keseluruhan sistem. Komponen fungsional dari sistem manajemen basis data dapat dibagi menjadi:

- The storage manager
- The query processor component
- The transaction management component



Gambar 1.6: Arsitektur DBMS pada mesin server terpusat

1.11 Database System

Sistem Basis Data mengacu pada organisasi komponen yang mendefinisikan dan mengatur pengumpulan, penyimpanan, pengelolaan, dan penggunaan data dalam lingkungan basis data.

Sistem basis data terdiri dari lima bagian utama:

- Hardware
- Software
- People
- Prosedur
- Data

1.12 Database Applications Architecture

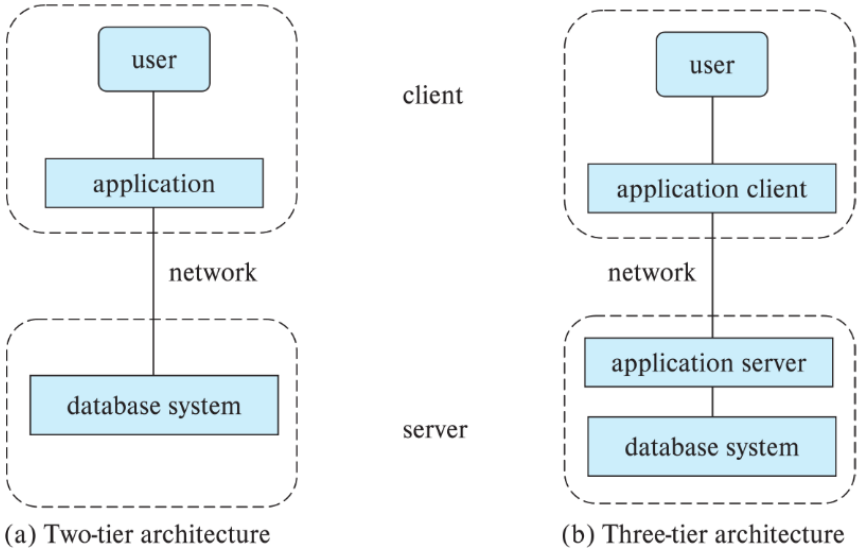
Dapat dibagi menjadi:

1. **Two-tier architecture**

   Aplikasi berada di mesin *client*, di mana ia memanggil fungsionalitas sistem basis data di server mesin.

2. **Two-tier architecture**

   Mesin *client* bertindak sebagai *front end* dan tidak mengandung panggilan basis data langsung



Gambar 1.7: Database Applications Architecture

1.13 Database Users

Dapat terdiri atas:

1. **Naive users** (tellers, agents, web users).

   Berinteraksi dengan sistem dengan **menggunakan** antarmuka pengguna yang telah ditentukan, seperti aplikasi web atau seluler. Pengguna naif juga dapat melihat laporan yang telah dibaca yang dihasilkan dari database.

2. **Application programmers**.

   Profesional komputer yang **menulis** program aplikasi. Pemrogram aplikasi dapat memilih dari banyak alat untuk mengembangkan antarmuka pengguna

3. **Sophisticated user** (analysts).

   Berinteraksi dengan sistem tanpa menulis program. Sebaliknya, mereka membentuk permintaan mereka dengan menggunakan bahasa permintaan database atau dengan menggunakan alat bantu seperti perangkat lunak analisis data.

4. **Database Administrator**.

   Menggunakan *adminstration tools* untuk melakukan *use* ke Basis Data.

Bab 2

Data Model

2.1 Data Modelling

Definisi 2.1: Basis Data

Sebuah teknik yang bertujuan untuk mengoptimalkan cara penyimpanan dan penggunaan informasi dalam sebuah organisasi

- Dimulai dengan identifikasi kelompok data utama, dan dilanjutkan dengan mendefinisikan isi detail dari masing-masing kelompok tersebut.
- Hasilnya: definisi terstruktur untuk semua informasi yang disimpan dan digunakan dalam suatu sistem tertentu.

Prasyarat penting untuk tahap analisis, desain, pemeliharaan & dokumentasi, serta peningkatan kinerja dari sistem yang sudah ada.

Terdapat tiga jenis data model yang berbeda yang dihasilkan selama proses dari tahap kebutuhan hingga ke database yang sebenarnya,

1. **Conceptual data model:**

   Sekumpulan spesifikasi data yang bersifat independen terhadap teknologi dan digunakan untuk mendiskusikan kebutuhan awal dengan para pemangku kepentingan bisnis.

2. **Logical data model:**

   Struktur data yang dapat diimplementasikan dalam database.

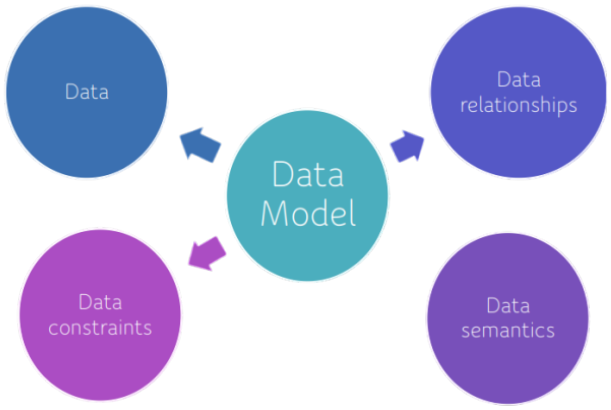
3. **Physical data model:**

   Mengorganisasi data ke dalam tabel, serta mempertimbangkan detail akses, performa, dan penyimpanan.

2.2 Data Model

Definisi 2.2

Kumpulan alat bantu untuk mendeskripsikan pada gambar berikut



Gambar 2.1: Data Model

Tipe-tipe data model:

- Entity Relationship Model
- Relational Model
- Object-based data model
- Hierarical Model
- Semi-structured data model (XML)
- Network Model

2.3 Relational Model

Konsep

Terdiri dari kumpulan tabel untuk merepresentasikan data dan hubungan antar data tersebut. Contoh: data tabular dalam relational model

customer_id	*customer_name*	*customer_street*	*customer_city*	*account_number*
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-101
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-201
677-89-9011	Hayes	3 Main St.	Harrison	A-102
182-73-6091	Turner	123 Putnam St.	Stamford	A-305
321-12-3123	Jones	100 Main St.	Harrison	A-217
336-66-9999	Lindsay	175 Park Ave.	Pittsfield	A-222
019-28-3746	Smith	72 North St.	Rye	A-201

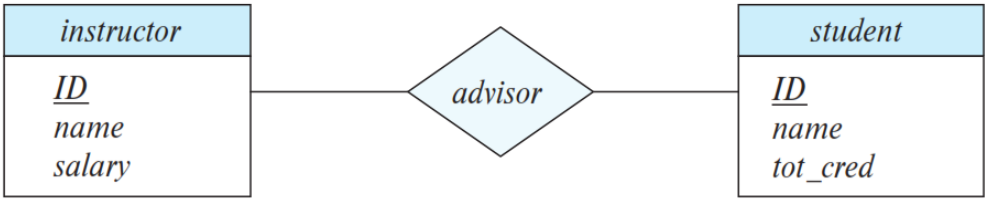
Gambar 2.2: Relational Model

2.4 Entity Relationship Model

Konsep

Banyak digunakan untuk desain basis data

- Desain basis data dalam model E-R biasanya dikonversi ke dalam desain dalam Model Relasional
- Memodelkan perusahaan sebagai kumpulan entitas dan hubungan
  - Entitas: sebuah “hal” atau “objek” dalam perusahaan yang yang dapat dibedakan dari objek lain
  - Relasi: hubungan di antara beberapa entitas
- Direpresentasikan secara diagramatis oleh *entity-relationship diagram (ERD)*



Gambar 2.3: Entity-Relationship Diagram

2.5 Object-Oriented Model

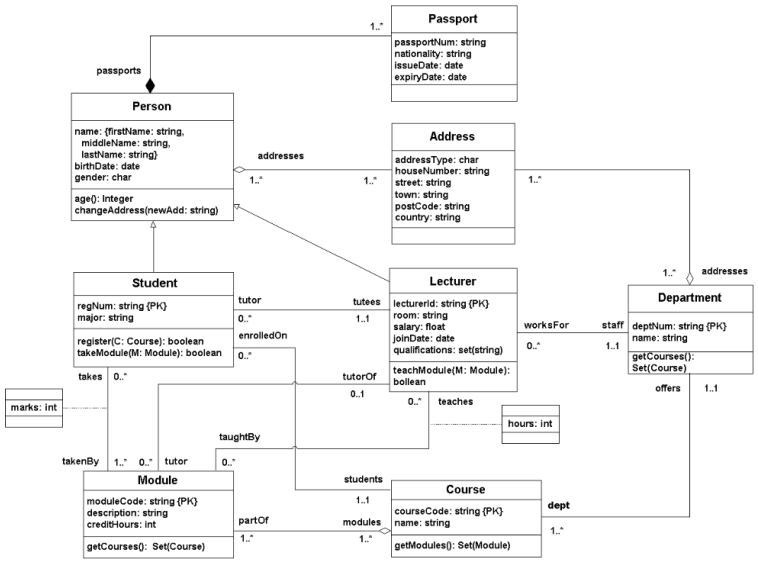
Konsep

Adaptasi dari paradigma pemrograman object-oriented (misalnya Smalltalk, C++) ke dalam sistem database.

Paradigma object-oriented didasarkan pada pengkapsulan kode dan data yang terkait dengan suatu objek ke dalam satu kesatuan unit.

Sebuah object memiliki hal-hal berikut yang terkait dengannya:

- Sekumpulan **variable** yang menyimpan data untuk objek tersebut. Nilai dari setiap variable itu sendiri adalah sebuah objek.
- Sekumpulan **message** yang dapat direspons oleh objek; setiap message dapat memiliki nol, satu, atau lebih parameter.
- Sekumpulan **method**, yang masing-masing merupakan blok kode untuk mengimplementasikan suatu message; sebuah method mengembalikan nilai sebagai respons terhadap message tersebut.



Gambar 2.4: Object-Oriented Model

2.6 Object-Relational Model

Konsep

Memperluas relational data model dengan menyertakan object orientation dan sistem tipe yang lebih kaya, termasuk tipe koleksi

- Object orientation menyediakan inheritance dengan subtype dan subtable
- Collection types mencakup nested relation, set, multiset, dan array
- Tetap mempertahankan dasar-dasar relasional, khususnya akses data yang bersifat declarative, sambil memperluas kemampuan dalam pemodelan

2.7 Semi-Structured Model (ex: XML)

Konsep

Kemampuan untuk menentukan tag baru, dan membuat struktur tag yang bersarang menjadikan XML sebagai cara yang sangat baik untuk bertukar data, bukan hanya dokumen

- XML telah menjadi dasar bagi semua format pertukaran data generasi baru
- Tersedia berbagai macam alat untuk parsing, browsing, dan querying dokumen/data XML

Karakteristik:

- Terorganisasi dalam entitas semantik
- Entitas yang serupa dikelompokkan bersama
- Entitas dalam kelompok yang sama mungkin tidak memiliki atribut yang sama
- Urutan atribut tidak harus penting
- Tidak semua atribut harus ada
- Ukuran dari atribut yang sama dalam suatu kelompok bisa berbeda
- Tipe dari atribut yang sama dalam suatu kelompok bisa berbeda
- Self-describing, data tidak teratur, dan tidak memiliki struktur awal (no a priori structure)

```
<bank>
   <account>
      <account-number>A-101</account-number>
      <branch-name>Downtown</branch-name>
      <balance>500</balance>
   </account>
   …
   <customer>
      <customer-name>Johnson</customer-name>
      <customer-street>Alma</customer-street>
      <customer-city>Palo Alto</customer-city>
   </customer>
   …
</bank>
```

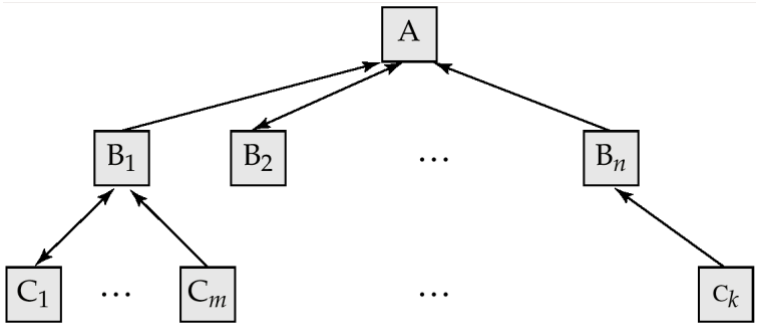
Gambar 2.5: Semi-Structure Data Model. Contoh: XML

2.8 Hierarical Model

Konsep

Sistem database hierarkis mainframe terkemuka di dunia pada tahun 1970-an dan awal 1980-an

- Skema database direpresentasikan sebagai kumpulan diagram struktur pohon
  - **a.** Hanya terdapat satu instance dari sebuah pohon database
  - **b.** Akar dari pohon tersebut adalah dummy node
  - **c.** Anak-anak dari node tersebut adalah instance nyata dari record type yang sesuai
- Skema untuk hierarchical database terdiri dari:
  - **a.** Kotak (boxes), yang merepresentasikan record types
  - **b.** Garis (lines), yang merepresentasikan link antar record
- Record types diorganisasikan dalam bentuk rooted tree



Gambar 2.6: Hierarical Model - seorang parent dapat memiliki panah yang menunjuk ke child, tetapi seorang child harus memiliki panah yang menunjuk ke parent-nya.

2.9 Network Model

Konsep

Data direpresentasikan sebagai kumpulan record

- Mirip dengan entity dalam E-R model
- Record dan field-nya direpresentasikan sebagai record type

Hubungan antar data direpresentasikan sebagai link

- Mirip dengan bentuk terbatas (biner) dari E-R relationship
- Pembatasan pada link bergantung pada apakah hubungan tersebut bersifat many-many, many-to-one, atau one-to-one

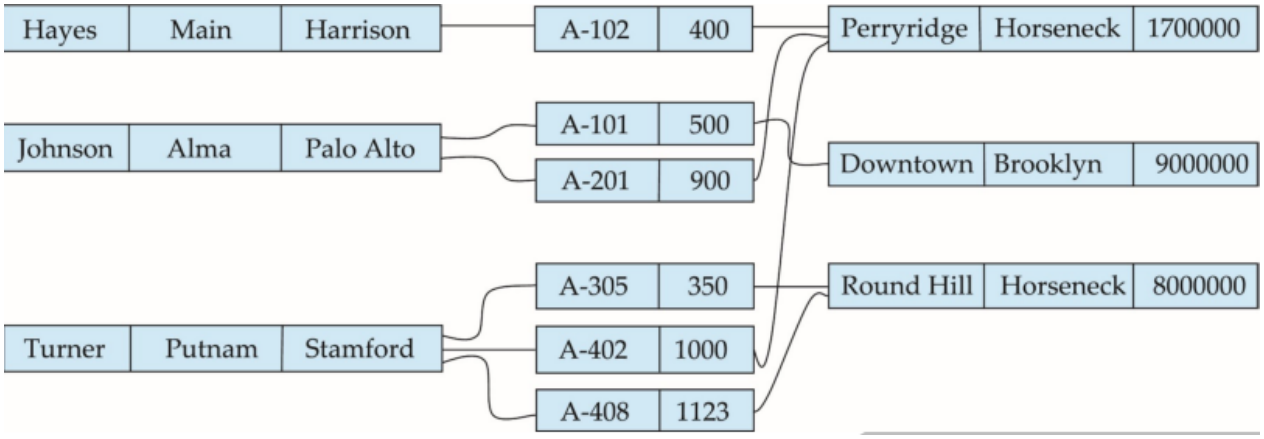
Struktur seperti graph

- Mirip dengan bentuk terbatas (biner) dari E-R relationship
- Pembatasan pada link bergantung pada apakah hubungan tersebut bersifat many-many, many-to-one, atau one-to-one

Diagram struktur data terdiri dari dua komponen dasar:

- Kotak (boxes), yang merepresentasikan record types
- Garis (lines), yang merepresentasikan link

Menentukan struktur logis keseluruhan dari database



Gambar 2.7: Network Model

Bab 3

Relational Model

3.1 Konsep Relational Model

Konsep

Relation Model adalah konsep matematis yang didasarkan pada ide mengenai set

- Model ini pertama kali diusulkan oleh Dr. E.F. Codd dari IBM Research pada tahun 1970 dalam makalah berikut:
- "A Relational Model for Large Shared Data Banks," Communications of the ACM, Juni 1970
- Makalah tersebut memicu revolusi besar dalam bidang database management dan membuat Dr. Codd meraih penghargaan bergengsi ACM Turing Award
- Mengapa mempelajari ini? Karena ini adalah model yang paling banyak digunakan
- Saat ini, RDBMS telah menjadi teknologi dominan dalam pengelolaan database, dan ada ratusan produk RDBMS untuk berbagai jenis komputer, mulai dari smartphone, PC, hingga mainframe

3.2 Relation

Definisi 3.1

Sebuah **relation** adalah tabel dua dimensi yang diberi nama dan berisi data. Tabel terdiri dari baris (record) dan kolom (attribute atau field).

Syarat agar sebuah tabel memenuhi syarat sebagai relation:

- Harus memiliki nama yang unik
- Setiap nilai atribut harus atomik (tidak multivalued, tidak komposit)
- Setiap baris harus unik (tidak boleh ada dua baris yang memiliki nilai yang sama persis di semua field)
- Atribut (kolom) dalam tabel harus memiliki nama yang unik
- Urutan kolom boleh tidak berurutan
- Urutan baris juga boleh tidak berurutan

Catatan: Semua relation berada dalam First Normal Form (1NF).

3.3 Attribute

Definisi 3.2

Setiap **attribute** dari sebuah *relation* memiliki nama. Sekumpulan nilai yang diperbolehkan untuk suatu atribut disebut sebagai **domain** dari atribut tersebut.

Karakteristik nilai atribut:

- Nilai atribut umumnya harus bersifat **atomik** (tidak dapat dibagi lagi)
- Contoh: nilai atribut dapat berupa nomor rekening, namun tidak boleh berupa himpunan nomor rekening

- Nilai khusus **null** merupakan anggota dari setiap domain
- Nilai null menyebabkan kerumitan dalam definisi banyak operasi
- Dalam pembahasan utama ini, efek dari nilai null diabaikan dan akan dibahas kemudian

Makna dari nilai NULL:

- Nilai tidak diketahui
- Nilai ada, tetapi tidak tersedia
- Atribut tidak berlaku untuk tuple ini (*value undefined*)

Penting: NULL $\neq$ NULL

3.4 Relation Schema & Instance

Definisi 3.3

- $A_1, A_2, \ldots, A_n$ adalah *attributes*
- $R = (A_1, A_2, \ldots, A_n)$ adalah *relation schema*

Contoh:

```
instructor = (ID, name, dept_name, salary)
```

- Nilai-nilai terkini dari suatu *relation* ditentukan oleh sebuah tabel
- Sebuah elemen t dari relation r disebut sebagai *tuple* dan direpresentasikan sebagai sebuah *row* dalam tabel

3.5 Database Schema

Definisi 3.4

Database schema adalah struktur logis dari sebuah database.
Database instance adalah snapshot dari data di dalam database pada suatu waktu tertentu.

Contoh:

- **Schema:** `instructor (ID, name, dept_name, salary)`
- **Instance:**

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

3.6 Why Split Information Across Relations?

Definisi 3.5

Menyimpan seluruh informasi dalam satu *relation* seperti:
`bank(account_number, balance, customer_name, ...)`
dapat menyebabkan beberapa masalah sebagai berikut:

- **Duplikasi informasi**

  Contoh: jika dua pelanggan memiliki satu akun yang sama (apa yang terduplikasi?)

- **Kebutuhan akan nilai *null***

  Contoh: untuk merepresentasikan pelanggan yang tidak memiliki akun

Normalization theory (yang akan dibahas nanti) digunakan untuk merancang skema relasional yang baik.

3.7 Keys

Definisi 3.6

Keys adalah atribut atau kombinasi atribut yang digunakan untuk mengidentifikasi *tuple* secara unik dalam suatu relasi.

- **Superkey** Himpunan satu atau lebih atribut yang dapat digunakan untuk mengidentifikasi tuple secara unik dalam suatu relasi.
- **Candidate Key** Superkey minimal, yaitu superkey yang tidak memiliki atribut berlebih. Setiap relasi bisa memiliki lebih dari satu candidate key.
- **Primary Key** Salah satu candidate key yang dipilih sebagai pengenal utama. Tidak boleh memiliki nilai `NULL`.
- **Foreign Key** Atribut dalam suatu relasi yang menjadi kunci acuan dari **primary key** di relasi lain. Digunakan untuk merepresentasikan relasi antar tabel.

3.8 Superkeys

Definisi 3.7

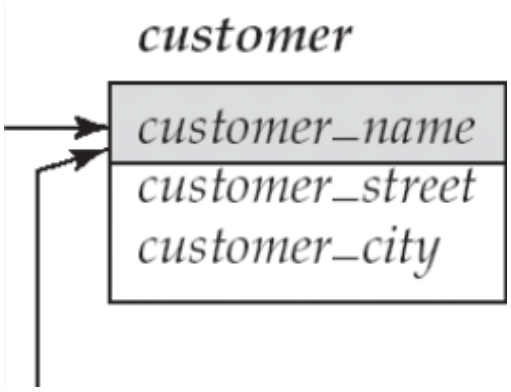
Misalkan $K \subseteq R$
K adalah **superkey** dari R jika nilai-nilai dari K cukup untuk mengidentifikasi sebuah *tuple* secara unik pada setiap kemungkinan *relation* $r(R)$.

Catatan: "kemungkinan r" berarti sebuah relasi r yang dapat eksis dalam model yang sedang kita buat.

- {customer_name, customer_street}
- {customer_name}

Kedua himpunan di atas adalah **superkey** dari `Customer`, jika tidak ada dua pelanggan yang memiliki nama yang sama.

Dalam praktik nyata, atribut seperti `customer_id` akan digunakan sebagai pengenal unik pelanggan, namun dalam contoh ini kita menggunakan nama pelanggan agar contoh tetap sederhana dengan asumsi bahwa nama pelanggan bersifat unik.



Gambar 3.1: Example Relation

3.9 Candidate & Primary Key

Definisi 3.8

K adalah **candidate key** jika K bersifat minimal.
Contoh: `{customer_name}` adalah candidate key untuk `Customer`, karena merupakan *superkey* dan tidak ada subset dari K yang juga superkey.

Definisi 3.9

Primary key adalah sebuah *candidate key* yang dipilih sebagai sarana utama untuk mengidentifikasi *tuple* dalam suatu relasi.

- Sebaiknya dipilih atribut yang nilainya tidak pernah atau sangat jarang berubah
- Contoh: email address unik, tetapi bisa saja berubah

Kunci bisa berupa:

- **Simple key**: satu atribut (single field)
- **Composite key**: lebih dari satu atribut (multi-field)

Key umumnya digunakan sebagai indeks untuk mempercepat respon terhadap query pengguna.

3.10 Foreign Key

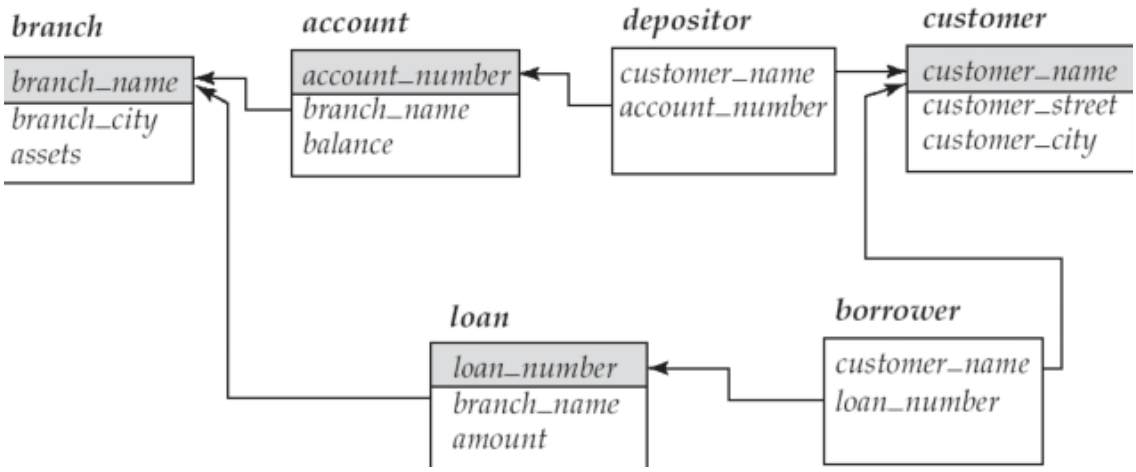
Definisi 3.10

Sebuah *relation schema* dapat memiliki atribut yang sesuai dengan **primary key** dari relasi lain. Atribut ini disebut sebagai **foreign key**.

Contoh:

- `customer_name` dan `account_number` pada relasi `depositor` merupakan foreign key ke relasi `customer` dan `account`.

Ketentuan: Hanya nilai-nilai yang terdapat dalam atribut **primary key** dari *referenced relation* yang boleh muncul di atribut **foreign key** pada *referencing relation.*



Gambar 3.2: Foreign Key in Red Circle

Contoh skema relasi dengan *primary key* dan *foreign key*:

- `branch = (branch_name, branch_city, assets)`
- `account = (account_number, branch_name, balance)`
- `customer = (customer_name, customer_street, customer_city)`
- `loan = (loan_number, branch_name, amount)`
- `depositor = (customer_name, account_number)`
- `borrower = (customer_name, loan_number)`

Foreign keys:

- `account(branch_name)` → `branch(branch_name)`
- `loan(branch_name)` → `branch(branch_name)`
- `depositor(customer_name)` → `customer(customer_name)`
- `depositor(account_number)` → `account(account_number)`
- `borrower(customer_name)` → `customer(customer_name)`
- `borrower(loan_number)` → `loan(loan_number)`

Note: `underlined` menunjukkan **primary key**

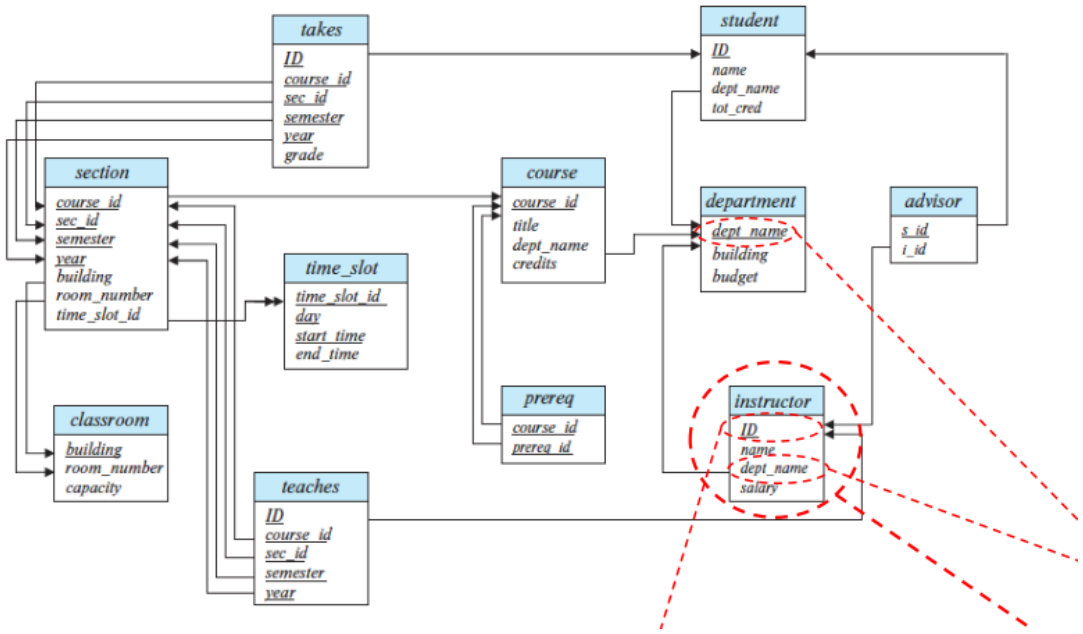
3.11 Schema Diagram

Definisi 3.11

Diagram skema (*schema diagram*) menggambarkan struktur logis dari database dalam bentuk diagram relasi.

Karakteristik dari schema diagram:

- Setiap **relation** direpresentasikan sebagai **kotak**
- Nama relasi ditulis di bagian atas (berwarna biru)
- Atribut-atribut ditampilkan di dalam kotak
- **Primary key** ditunjukkan dengan garis bawah pada atribut
- **Foreign key constraints** digambarkan sebagai **panah** dari atribut foreign key ke atribut primary key pada relasi yang dirujuk



Gambar 3.3: Schema Diagram

Contoh (dari gambar):

- Relasi `instructor` memiliki `ID` sebagai primary key, dan `dept_name` sebagai foreign key ke `department`
- Relasi `teaches` memiliki `ID`, `course_id`, `sec_id`, `semester`, dan `year` sebagai kunci gabungan

Diagram ini sangat membantu dalam memahami struktur dan hubungan antar tabel dalam skema database relasional.

3.12 Integrity Constraint

Definisi 3.12

Integrity constraint adalah aturan-aturan yang digunakan untuk menjaga konsistensi dan validitas data dalam database relasional.

Tiga jenis utama integrity constraints:

- **Domain Constraints** Semua nilai dalam suatu kolom (atribut) harus berasal dari domain yang sama.
- **Entity Integrity** Aturan ini memastikan bahwa setiap relasi memiliki **primary key** dan bahwa semua nilai pada atribut primary key tersebut adalah **valid** (tidak boleh bernilai `NULL`).
- **Referential Integrity** Aturan yang menjaga konsistensi antar baris dari dua relasi yang berbeda melalui penggunaan **foreign key**.

3.12.1 Domain Constraint

Definisi 3.13

Domain adalah himpunan nilai yang valid untuk suatu atribut dalam relasi. Setiap atribut dalam relasi harus memiliki domain yang telah ditentukan.

Sebuah definisi domain biasanya terdiri dari komponen-komponen berikut:

- Nama domain (*domain name*)
- Makna atau deskripsi domain
- Tipe data (*data type*)
- Ukuran atau panjang (*size/length*)
- Nilai-nilai yang diperbolehkan atau rentang nilai yang diperbolehkan (jika berlaku)

TABLE 4-1 Domain Definitions for INVOICE Attributes

Attribute	Domain Name	Description	Domain
CustomerID	Customer IDs	Set of all possible customer IDs	character: size 5
CustomerName	Customer Names	Set of all possible customer names	character: size 25
CustomerAddress	Customer Addresses	Set of all possible customer addresses	character: size 30
CustomerCity	Cities	Set of all possible cities	character: size 20
CustomerState	States	Set of all possible states	character: size 2
CustomerPostalCode	Postal Codes	Set of all possible postal zip codes	character: size 10
OrderID	Order IDs	Set of all possible order IDs	character: size 5
OrderDate	Order Dates	Set of all possible order dates	date: format mm/dd/yy
ProductID	Product IDs	Set of all possible product IDs	character: size 5
ProductDescription	Product Descriptions	Set of all possible product descriptions	character: size 25
ProductFinish	Product Finishes	Set of all possible product finishes	character: size 15
ProductStandardPrice	Unit Prices	Set of all possible unit prices	monetary: 6 digits
ProductLineID	Product Line IDs	Set of all possible product line IDs	integer: 3 digits
OrderedQuantity	Quantities	Set of all possible ordered quantities	integer: 3 digits

Gambar 3.4: Domain Constraint

3.12.2 Entity Integrity

Definisi 3.14

Entity Integrity adalah aturan yang menyatakan bahwa tidak ada atribut **primary key** (atau komponen dari *composite primary key*) yang boleh bernilai `NULL`.

Contoh:

- Jika kita memilih `CustomerID` sebagai **primary key**, maka setiap baris data harus memiliki nilai `CustomerID` yang valid dan **tidak boleh kosong**.

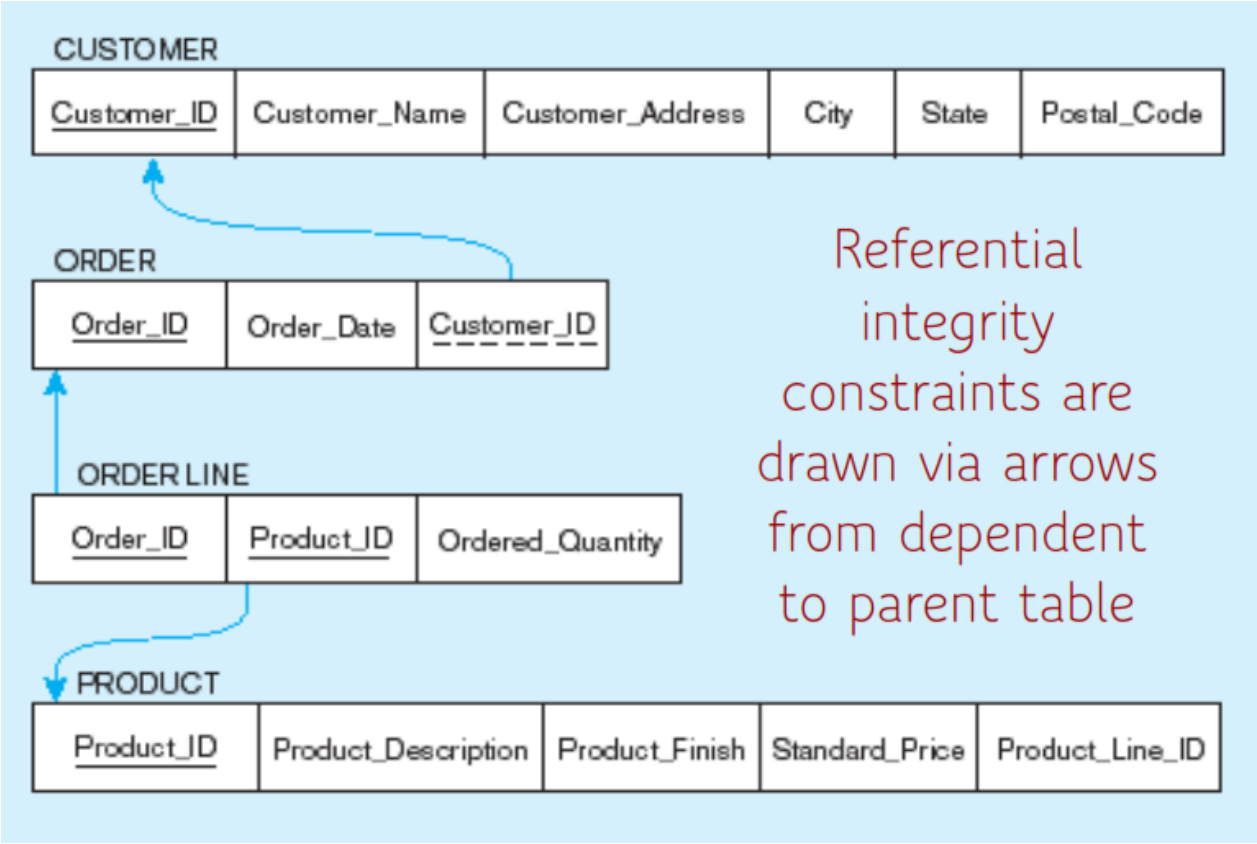
3.12.3 Referential Integrity

Definisi 3.15

Referential Integrity adalah aturan yang menyatakan bahwa setiap nilai **foreign key harus cocok** dengan nilai **primary key** pada relasi yang dirujuk (*referenced relation*) **atau** foreign key tersebut boleh bernilai `NULL`.

Contoh: Delete Rules

- **Restrict** – Tidak mengizinkan penghapusan pada sisi "parent" jika masih ada baris terkait di sisi "dependent"
- **Cascade** – Secara otomatis menghapus baris di sisi "dependent" yang berkaitan dengan baris sisi "parent" yang dihapus
- **Set-to-Null** – Mengatur nilai foreign key menjadi `NULL` pada sisi dependent jika entri pada parent dihapus (tidak berlaku untuk entitas lemah)



Gambar 3.5: Referential Integrity

Bab 4

SQL (Part 1)

4.1 History of SQL

Definisi 4.1

Bahasa **Sequel** dikembangkan oleh IBM sebagai bagian dari proyek *System R* di IBM San Jose Research Laboratory. Kemudian diubah namanya menjadi **Structured Query Language (SQL)**.

Standar ANSI dan ISO untuk SQL:

- SQL-86
- SQL-89
- SQL-92
- SQL:1999 (bahasa ini menjadi *Y2K compliant*)
- SQL:2003

Sistem komersial umumnya menyediakan sebagian besar, jika tidak semua, fitur dari SQL-92, ditambah fitur tambahan dari standar-standar berikutnya dan fitur khusus masing-masing sistem. Tidak semua contoh SQL mungkin dapat dijalankan di setiap sistem basis data.

4.2 SQL Parts

- **DML** – Menyediakan kemampuan untuk melakukan query terhadap data di database, serta untuk melakukan operasi `insert`, `delete`, dan `update` terhadap tuple.
- **Integrity** – Bagian dari **DDL** yang berisi perintah untuk menentukan *integrity constraints*.
- **View Definition** – Bagian dari **DDL** yang berisi perintah untuk mendefinisikan *view*.
- **Transaction Control** – Berisi perintah untuk menentukan awal dan akhir dari suatu transaksi.
- **Embedded SQL dan Dynamic SQL** – Mendefinisikan bagaimana perintah SQL dapat disisipkan dalam bahasa pemrograman umum (*general-purpose programming languages*).
- **Authorization** – Berisi perintah untuk menentukan hak akses terhadap relasi dan *views*.

4.3 Basic Query Structure

Struktur umum dari sebuah perintah SQL adalah sebagai berikut:

```
select A1, A2, ..., An
from   r1, r2, ..., rm
where  P
```

Keterangan:

- A_i merepresentasikan sebuah **attribute**
- r_i merepresentasikan sebuah **relation**
- P adalah sebuah **predicate**

Hasil dari sebuah query SQL adalah sebuah **relation**.

4.4 The `select` Clause

Konsep

Klausa `select` digunakan untuk menentukan atribut-atribut yang diinginkan dalam hasil sebuah query. Klausa ini berkorespondensi dengan operasi *projection* dalam aljabar relasional.

Contoh: Menemukan nama semua `instructor`:

```
select name
from instructor
```

Catatan: Nama dalam SQL bersifat **case-insensitive**, artinya:

- `Name` ≡ `NAME` ≡ `name`
- Beberapa orang menggunakan huruf kapital untuk elemen yang dianggap penting (misalnya keyword SQL)

Konsep

SQL mengizinkan adanya data duplikat baik dalam relasi maupun hasil query.

Untuk menghilangkan duplikat, gunakan kata kunci `distinct` setelah `select`.

Contoh: Menemukan nama-nama departemen dari semua `instructor`, tanpa duplikat

```
select distinct dept_name
from instructor
```

Kata kunci `all` menyatakan bahwa duplikat tidak dihilangkan (ini adalah default-nya SQL).

```
select all dept_name
from instructor
```

Konsep

Simbol `*` dalam klausa `select` menyatakan bahwa seluruh atribut akan ditampilkan.

Contoh: Menampilkan `instructor` dengan seluruh attributnya:

```
select *
from instructor
```

Atribut juga bisa berupa literal tanpa `from` clause:

```
select '437'
```

Hasilnya adalah tabel dengan satu kolom dan satu baris berisi nilai "437"

Kolom dapat diberi nama menggunakan alias:

```
select '437' as FOO
```

Literal juga bisa digunakan bersama `from` clause:

```
select 'A'
from instructor
```

Hasilnya adalah tabel dengan satu kolom dan N baris (di mana N adalah jumlah tuple pada tabel `instructor`), dengan setiap baris berisi nilai "A"

Konsep

Klausa `select` dapat berisi ekspresi aritmatika yang melibatkan operasi `+`, `-`, `*`, dan `/`, serta dapat diaplikasikan pada konstanta maupun atribut dari tuple.

Contoh query:

```
select ID, name, salary/12
from instructor
```

akan menghasilkan relasi yang sama dengan relasi `instructor`, kecuali nilai atribut `salary` dibagi 12.

Kita juga bisa memberikan nama pada hasil ekspresi tersebut menggunakan klausa `as`:

```
select ID, name, salary/12 as monthly_salary
```

4.5 The `where` Clause

Konsep

Klausa `where` menetapkan kondisi yang harus dipenuhi oleh hasil query. Berkorespondensi dengan *selection predicate* dalam aljabar relasional.

Contoh: Mencari semua `instructor` dari departemen `Comp. Sci.`

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

SQL mendukung penggunaan operator logika seperti `and`, `or`, dan `not`. Operand dari operator ini dapat berupa ekspresi yang melibatkan operator perbandingan:

- <, <=, >, >=, =, <>
- Dapat diterapkan pada hasil dari ekspresi aritmatika

Contoh: Mencari semua `instructor` di `Comp. Sci.` dengan gaji lebih dari 70000:

```
select name
from instructor
where dept_name = 'Comp. Sci.' and salary > 70000
```

4.6 The `from` Clause

Konsep

Klausa `from` menyebutkan relasi-relasi yang terlibat dalam sebuah query. Berkorespondensi dengan operasi *Cartesian product* dalam aljabar relasional.

Contoh: Mencari hasil produk Kartesius antara `instructor` dan `teaches`

```
select *
from instructor, teaches
```

Penjelasan:

- Menghasilkan semua kemungkinan pasangan `instructor`–`teaches`, dengan seluruh atribut dari kedua relasi.
- Jika ada atribut yang sama (misalnya `ID`), maka hasilnya akan menggunakan nama relasi sebagai prefix, seperti `instructor.ID`.

Produk Kartesian biasanya tidak terlalu berguna secara langsung, tetapi menjadi berguna bila dikombinasikan dengan `where`-clause untuk menyaring hasil (*selection operation* dalam aljabar relasional).

Contoh 1: Menemukan nama semua `instructor` yang mengajar suatu mata kuliah beserta `course_id`-nya

```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID
```

Contoh 2: Menemukan nama semua `instructor` dari departemen `Art` yang mengajar suatu mata kuliah beserta `course_id`-nya

```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID
  and instructor.dept_name = 'Art'
```

4.7 The Rename Operation

Konsep

SQL memungkinkan untuk mengganti nama relasi atau atribut menggunakan klausa `as`:

```
old-name as new-name
```

Contoh: Menemukan nama semua `instructor` yang memiliki gaji lebih tinggi daripada salah satu `instructor` di departemen `'Comp. Sci.'`

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary
  and S.dept_name = 'Comp. Sci.'
```

Kata kunci `as` bersifat opsional dan dapat dihilangkan:

`instructor as T` ≡ `instructor T`

4.8 String Operations

Konsep

SQL menyediakan operator pencocokan string untuk melakukan perbandingan terhadap data karakter. Operator `like` menggunakan pola yang dideskripsikan menggunakan dua karakter spesial:

- `%` mencocokkan **substring** apa pun
- `_` mencocokkan **satu karakter** apa pun

Contoh: Menemukan semua `instructor` yang memiliki substring `dar` di namanya:

```
select name
from instructor
where name like '%dar%'
```

Untuk mencocokkan string literal seperti "100

```
like '100 \%' escape '\'
```

Pada contoh di atas, kita menggunakan \ (\) sebagai karakter pelolos (*escape character*).

Catatan: Pola pencocokan (pattern matching) bersifat **case-sensitive**.

Contoh pattern matching:

- `'Intro%'` mencocokkan string apa pun yang diawali dengan `Intro`
- `'%Comp%'` mencocokkan string apa pun yang mengandung `Comp` sebagai substring
- `'_ _ _'` mencocokkan string dengan tepat tiga karakter
- `' _ _ _ %'` mencocokkan string dengan minimal tiga karakter

SQL juga mendukung berbagai operasi string lainnya, seperti:

- **Konkatenasi string** menggunakan `||`
- **Konversi huruf kapital/kecil** antar upper/lower case
- **Fungsi manipulasi string** seperti mencari panjang string, ekstrak substring, dll.

4.9 Ordering the Display of Tuples

Menampilkan nama semua `instructor` dalam urutan alfabet:

```
select distinct name
from instructor
order by name
```

- Gunakan `desc` untuk urutan menurun, atau `asc` untuk urutan menaik (default).
- Contoh: `order by name desc`
- Dapat mengurutkan berdasarkan beberapa atribut:

  ```
  order by dept_name, name
  ```

4.10 Where Clause Predicates

Konsep

SQL mendukung operator `between` untuk perbandingan range nilai.

Contoh: Mencari semua `instructor` dengan gaji antara $90.000 dan $100.000:

```
select name
from instructor
where salary between 90000 and 100000
```

Tuple comparison:

```
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, 'Biology')
```

4.11 Set Operations

- Cari mata kuliah yang dijalankan pada Fall 2017 atau Spring 2018:

```
(select course_id from section where sem = 'Fall' and year = 2017)
union
(select course_id from section where sem = 'Spring' and year = 2018)
```

- Cari mata kuliah yang dijalankan pada Fall 2017 dan Spring 2018:

```
(select course_id from section where sem = 'Fall' and year = 2017)
intersect
(select course_id from section where sem = 'Spring' and year = 2018)
```

- Cari mata kuliah yang dijalankan pada Fall 2017 tetapi tidak di Spring 2018:

```
(select course_id from section where sem = 'Fall' and year = 2017)
except
(select course_id from section where sem = 'Spring' and year = 2018)
```

Operasi `union`, `intersect`, dan `except` secara default **menghapus duplikat**.

Untuk mempertahankan semua duplikat, gunakan:

- `union all`
- `intersect all`
- `except all`

4.12 Null Values

Konsep

- Tuple dapat memiliki nilai `null` pada beberapa atributnya.
- `null` menyatakan nilai yang tidak diketahui atau tidak ada.
- Ekspresi aritmatika yang melibatkan `null` akan menghasilkan `null`.

  Contoh: `5 + null` menghasilkan `null`

Gunakan `is null` untuk mengecek apakah suatu atribut bernilai `null`:

```
select name
from instructor
where salary is null
```

Gunakan `is not null` untuk memeriksa atribut yang tidak null.

4.12.1 Nulls in Comparison and Boolean Logic

- SQL menganggap hasil dari perbandingan dengan `null` sebagai **unknown**.

  Contoh: `5 < null`, `null <> null`, `null = null` → `unknown`

- Dalam `where clause`, ekspresi logika bisa melibatkan: `and`, `or`, dan `not`, yang perlu diperluas untuk menangani nilai `unknown`.

  Ekspansi logika boolean:

  AND

  `true AND unknown` = `unknown`

  `false AND unknown` = `false`

  `unknown AND unknown` = `unknown`

  OR

  `unknown OR true` = `true`

  `unknown OR false` = `unknown`

  `unknown OR unknown` = `unknown`

Result of `where` clause predicate is treated as `false` if it evaluates to `unknown`

Bab 5

SQL (Part 2)

5.1 Aggregate Functions

Definisi 5.1

Fungsi-fungsi yang digunakan untuk menghitung nilai dari multiset pada kolom suatu relation:

- `avg`: nilai rata-rata
- `min`: nilai minimum
- `max`: nilai maksimum
- `sum`: jumlah total
- `count`: jumlah elemen

Examples

- Rata-rata gaji dosen di departemen Computer Science:

```
select avg(salary)
from instructor
where dept_name = 'Comp. Sci.';
```

- Jumlah total instruktur yang mengajar di Spring 2018:

```
select count(distinct ID)
from teaches
where semester = 'Spring' and year = 2018;
```

- Jumlah tuple pada relation `course`:

```
select count(*)
from course;
```

5.1.1 Group By

Konsep

`Group by` mengelompokkan tuple berdasarkan nilai atribut tertentu. Umumnya digunakan bersama fungsi agregat.

```
select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name;
```

Note: Semua atribut di `select` yang bukan fungsi agregat harus ada di `group by`. Contoh salah:

```
select dept_name, ID, avg(salary)
from instructor
group by dept_name;  -- ERROR
```

5.1.2 Having Clause

Konsep

`having` digunakan untuk memberikan kondisi pada hasil agregasi. Karena `where` tidak bisa digunakan dengan fungsi agregat.

```
select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name
having avg(salary) > 42000;
```

Catatan:

- `where` digunakan sebelum `group by`.
- `having` digunakan setelah `group by` dan setelah agregasi terbentuk.

5.2 Joined Relations

Konsep

Join operations mengambil dua *relation* dan menghasilkan satu *relation* baru sebagai hasilnya.

Operasi *join* merupakan hasil dari *Cartesian product* dua relasi, lalu diseleksi berdasarkan kecocokan nilai pada atribut-atribut tertentu (biasanya foreign key dan primary key). Operasi ini juga menentukan atribut mana yang akan muncul di hasil akhir.

Join biasanya digunakan sebagai ekspresi subquery pada klausa `from`.

- Natural Join
- Inner Join
- Outer Join

5.2.1 Natural Join in SQL

Konsep

Natural join mencocokkan tuple berdasarkan atribut-atribut yang memiliki nama sama di kedua relasi, dan menghilangkan duplikasi kolom pada hasil.

- Contoh tanpa *natural join*:

```
select name, course_id
from student, takes
where student.ID = takes.ID;
```

- Contoh dengan *natural join*:

```
select name, course_id
from student natural join takes;
```

- Natural join bisa digunakan lebih dari dua relasi:

```
select A1, A2, ..., An
from r1 natural join r2 natural join ... natural join rn
where P;
```

5.2.2 Bahaya dalam Penggunaan Natural Join

Konsep

Waspadai atribut tidak terkait namun memiliki nama sama. Hal ini dapat menyebabkan hasil *join* yang salah.

- Contoh benar:

```
select name, title
from student natural join takes, course
where takes.course_id = course.course_id;
```

- Contoh salah:

```
select name, title
from student natural join takes natural join course;
```

Query ini akan mengabaikan pasangan `(student name, course title)` ketika course diambil dari departemen berbeda dari departemen mahasiswa.

5.3 Outer Join

Konsep

Outer join adalah ekstensi dari operasi *join* yang mencegah kehilangan informasi.

- Melakukan join, lalu menambahkan tuple dari salah satu relasi yang tidak memiliki pasangan, dengan nilai `null` pada atribut relasi lainnya.
- Bentuk outer join:
  - Left outer join
  - Right outer join
  - Full outer join

5.3.1 Outer Join Examples

Definisi 5.2

Outer Join adalah jenis operasi *join* dalam SQL yang menggabungkan baris dari dua tabel dan menyertakan baris yang tidak memiliki pasangan yang cocok dari salah satu tabel.

Contoh relasi:

- **Relation course**

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- **Relation prereq**

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

Perhatikan bahwa:

- Informasi `course` tidak tersedia untuk `CS-347`.
- Informasi `prereq` tidak tersedia untuk `CS-315`.

5.3.2 Left Outer Join

Konsep

Left Outer Join menyertakan semua baris dari relasi kiri (course), dan baris dari relasi kanan (prereq) jika ada kecocokan. Jika tidak, nilai `NULL` digunakan.

course natural left outer join prereq

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

Dalam *relational algebra*: `course` $\overline{\bowtie}$ `prereq`

5.3.3 Right Outer Join

Konsep

Right Outer Join menyertakan semua baris dari relasi kanan (prereq), dan baris dari relasi kiri (course) jika ada kecocokan. Jika tidak, nilai `NULL` digunakan.

course natural right outer join prereq

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

Dalam *relational algebra*: `course` $\overline{\bowtie}$ `prereq`

5.3.4 Full Outer Join

Konsep

Full Outer Join menyertakan semua baris dari kedua relasi (course dan prereq). Baris yang tidak cocok akan menampilkan `NULL` pada bagian yang tidak terpenuhi.

```
course natural full outer join prereq
```

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

Dalam *relational algebra*: `course` ⟗ `prereq`

5.4 Joined Types and Conditions

Konsep

Join operations mengambil dua relasi dan menghasilkan relasi baru sebagai hasilnya.

Operasi-operasi tambahan ini biasanya digunakan sebagai ekspresi subquery di dalam klausa `from`.

Join condition – mendefinisikan tuple mana di dua relasi yang cocok:

- `natural`
- `on <predicate>`
- `using (A1, A2, ..., An)`

Join type – mendefinisikan bagaimana tuple yang tidak cocok ditangani:

- `inner join`
- `left outer join`
- `right outer join`
- `full outer join`

5.5 Joined Relations – Examples (1)

```
course natural right outer join prereq
```

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

```
course full outer join prereq using (course_id)
```

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

5.6 Joined Relations – Examples (2)

```
course inner join prereq on
course.course_id = prereq.course_id
```

course_id	title	dept_name	credits	prereq_id	course_id
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190

```
course left outer join prereq on
course.course_id = prereq.course_id
```

course_id	title	dept_name	credits	prereq_id	course_id
BIO-301	Genetics	Biology	4	BIO-101	BIO-301
CS-190	Game Design	Comp. Sci.	4	CS-101	CS-190
CS-315	Robotics	Comp. Sci.	3	*null*	*null*

Bab 6

SQL (Part 3)

6.1 Nested Subqueries

Konsep

SQL menyediakan mekanisme untuk menyusun **subquery**. Sebuah **subquery** adalah ekspresi `select-from-where` yang disisipkan di dalam query lain.

Struktur umum:

```
select A1, A2, ..., An
from r1, r2, ..., rm
where P
```

Penempatan Subquery:

- **Where clause:** P dapat digantikan dengan bentuk: B ⟨operation⟩ (subquery)
  B adalah atribut dan `operation` didefinisikan kemudian.
- **From clause:** r_i bisa diganti dengan subquery valid mana pun.
- **Select clause:** A_i bisa diganti dengan subquery yang menghasilkan satu nilai.

6.1.1 Nested Query in Where Clause

- **Set Membership (`IN`):** Mengecek apakah nilai atribut merupakan anggota dari hasil subquery.
- **Set Comparison (`<relational operator> SOME/ALL`):** Mengevaluasi apakah nilai atribut memenuhi kondisi terhadap beberapa atau semua hasil subquery.
- **Test of Empty Relation (`EXISTS`):** Menentukan apakah subquery menghasilkan baris (tuple) atau tidak.
- **Test for Absence of Duplicates (`UNIQUE`):** Mengecek apakah hasil dari subquery unik (tidak ada duplikat).

6.1.2 Set Membership Examples

Menemukan course yang ditawarkan di Fall 2017 dan Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year = 2017 and
      course_id in (select course_id
                    from section
                    where semester = 'Spring' and year = 2018);
```

Menemukan course yang ditawarkan di Fall 2017 tetapi tidak di Spring 2018

```
select distinct course_id
from section
where semester = 'Fall' and year = 2017 and
      course_id not in (select course_id
                        from section
                        where semester = 'Spring' and year = 2018);
```

6.1.3 More Subquery Examples

Menampilkan nama instruktur yang bukan "Mozart" maupun "Einstein":

```
select distinct name
from instructor
where name not in ('Mozart', 'Einstein');
```

Menghitung jumlah mahasiswa (unik) yang mengikuti course section yang diajarkan instruktur dengan ID 10101:

```
select count(distinct ID)
from takes
where (course_id, sec_id, semester, year) in
    (select course_id, sec_id, semester, year
     from teaches
     where teaches.ID = 10101);
```

Catatan: query di atas dapat ditulis lebih sederhana, namun disusun demikian untuk menunjukkan fitur SQL.

6.2 Set Comparison - `some` Clause

Konsep

some digunakan untuk membandingkan nilai terhadap *set* hasil dari subquery. Operasi bernilai `true` jika ekspresi bernilai benar terhadap setidaknya satu elemen.

Contoh: Temukan nama instruktur dengan gaji lebih besar dari *setidaknya satu* instruktur di departemen Biologi.

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Biology';
```

Versi dengan `> some`:

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept_name = 'Biology');
```

6.2.1 Definition of `some` Clause

Formal: F $\langle \text{comp} \rangle$ `some` $r \Leftrightarrow \exists t \in r$ such that $(F \langle \text{comp} \rangle t)$
Operator yang dapat digunakan: `<`, `<=`, `>`, `>=`, `=`, $\neq$

- $5 <$ `some` $\{0, 5, 6\} =$ true
- $5 <$ `some` $\{0, 5\} =$ false
- $5 =$ `some` $\{0, 5\} =$ true
- $5 \neq$ `some` $\{0, 5\} =$ true (karena $0 \neq 5$)

`(= some)` ekuivalen dengan `in`
`(≠ some)` tidak ekuivalen dengan `not in`

6.3 Set Comparison - `all` Clause

Konsep

all digunakan untuk membandingkan nilai terhadap *seluruh* hasil dari subquery. Operasi bernilai `true` jika ekspresi bernilai benar terhadap semua elemen.

Contoh: Temukan nama instruktur yang memiliki gaji lebih tinggi dari semua instruktur di departemen Biologi.

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept_name = 'Biology');
```

6.4 Definition of `all` Clause

Formal: $F\ \langle\text{comp}\rangle\ \texttt{all}\ r \Leftrightarrow \forall t \in r\ (F\ \langle\text{comp}\rangle\ t)$

- $5 < \texttt{all}\{5, 6\}$ = false
- $5 < \texttt{all}\{6, 10\}$ = true
- $5 = \texttt{all}\{4, 5\}$ = false
- $5 \neq \texttt{all}\{4, 6\}$ = true

$(\neq$ `all`) ekuivalen dengan `not in`
(`= all`) tidak ekuivalen dengan `in`

6.5 Test for Empty Relations

Definisi 6.1

EXISTS mengembalikan nilai `true` jika hasil subquery tidak kosong.

- `exists r` $\Leftrightarrow$ $r \neq \emptyset$
- `not exists r` $\Leftrightarrow$ $r = \emptyset$

6.5.1 Use of `exists` Clause

Konsep

`exists` digunakan untuk mengecek apakah subquery menghasilkan setidaknya satu baris (non-empty).

Contoh query untuk mencari *semua course yang ditawarkan di Fall 2017 dan Spring 2018*:

```
select course_id
from section as S
where semester = 'Fall' and year = 2017 and
      exists (select *
              from section as T
              where semester = 'Spring' and year = 2018
              and S.course_id = T.course_id);
```

Correlation name: variabel `S` pada query luar
Correlated subquery: subquery bagian dalam

6.6 Use of `not exists` Clause

Konsep

`not exists` menghasilkan `true` jika subquery tidak menghasilkan baris apa pun.

Contoh: *Temukan mahasiswa yang telah mengambil semua course yang ditawarkan oleh departemen Biologi.*

```
select distinct S.ID, S.name
from student as S
where not exists (
    (select course_id
     from course
     where dept_name = 'Biology')
     except
    (select T.course_id
     from takes as T
     where S.ID = T.ID));
```

- Query pertama: daftar course dari Biologi
- Query kedua: course yang diambil oleh mahasiswa tertentu

Catatan: Tidak dapat ditulis menggunakan `= all` dan turunannya.

6.7 Test for Absence of Duplicate Tuples

Konsep

`unique` digunakan untuk mengecek apakah hasil subquery memiliki **tidak ada** tuple yang duplikat.

Contoh: *Cari semua course yang hanya ditawarkan paling banyak sekali di tahun 2017.*

```
select T.course_id
from course as T
where unique (
    select R.course_id
    from section as R
    where T.course_id = R.course_id
          and R.year = 2017);
```

6.8 Subqueries in the `from` Clause

Konsep

Subquery bisa digunakan di dalam `from` clause sebagai relasi temporer.

Contoh: *Temukan rata-rata gaji instruktur untuk tiap departemen yang memiliki rata-rata di atas $42.000.*

```
select dept_name, avg_salary
from (select dept_name, avg(salary) as avg_salary
      from instructor
      group by dept_name)
where avg_salary > 42000;
```

Versi lain dengan `alias` eksplisit:

```
select dept_name, avg_salary
from (select dept_name, avg(salary)
      from instructor
      group by dept_name)
      as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```

6.9 WITH Clause

Definisi 6.2: WITH

`WITH` clause menyediakan cara untuk mendefinisikan relasi temporer yang hanya tersedia pada query tempat `WITH` tersebut digunakan.

Contoh: *Temukan semua departemen dengan anggaran maksimum*

```
with max_budget (value) as
    (select max(budget)
     from department)
select department.name
from department, max_budget
where department.budget = max_budget.value;
```

6.9.1 Complex Queries using `WITH` Clause

Tujuan: Temukan semua departemen yang total gajinya lebih besar dari rata-rata total gaji seluruh departemen.

```
with dept_total (dept_name, value) as
    (select dept_name, sum(salary)
     from instructor
     group by dept_name),
     dept_total_avg(value) as
    (select avg(value)
     from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value > dept_total_avg.value;
```

6.10 Scalar Subquery

Definisi 6.3: Scalar Subquery

Subquery yang digunakan pada konteks yang mengharapkan satu nilai (single value).

Contoh: *Daftarkan semua departemen beserta jumlah instruktur pada masing-masing*

```
select dept_name,
       (select count(*)
        from instructor
        where department.dept_name = instructor.dept_name)
       as num_instructors
from department;
```

Catatan: Akan terjadi *runtime error* jika subquery menghasilkan lebih dari satu baris.

Bab 7

SQL (Part 4)

7.1 Views

Konsep

Dalam beberapa kasus, tidak semua pengguna perlu melihat keseluruhan model logis (yaitu semua relasi yang sebenarnya disimpan dalam database).
View menyediakan mekanisme untuk menyembunyikan sebagian data dari pengguna tertentu.
Relasi apapun yang bukan bagian dari model konseptual, tetapi ditampilkan kepada pengguna sebagai *"virtual relation"*, disebut **view**.

Contoh: Seseorang hanya perlu mengetahui *name* dan *dept_name* dari instruktur tanpa melihat *salary*, maka cukup melihat:

```
select ID, name, dept_name
from instructor
```

7.1.1 View Definition

Definisi 7.1

View didefinisikan menggunakan perintah `create view` dengan format:

```
create view v as <query expression>
```

- `<query expression>` dapat berupa ekspresi SQL legal apa pun.
- Nama view diwakili oleh `v`.
- Setelah didefinisikan, view dapat digunakan seperti relasi biasa dalam query lain.
- View tidak menyimpan hasil query, tetapi menyimpan definisi query-nya.

7.1.2 View Usage

Contoh 1: View instruktur tanpa informasi gaji

```
create view faculty as
    select ID, name, dept_name
    from instructor;
```

Query: Cari instruktur dari departemen Biologi

```
select name
from faculty
where dept_name = 'Biology';
```

Contoh 2: View total gaji per departemen

```
create view departments_total_salary(dept_name, total_salary) as
    select dept_name, sum(salary)
    from instructor
    group by dept_name;
```

7.1.3 Views Defined Using Other Views

- Satu view bisa digunakan di dalam ekspresi pembentuk view lainnya.
- View v_1 dikatakan **depends directly** pada v_2 jika v_2 digunakan di dalam definisi v_1.
- View v_1 **depends on** v_2 jika:
  - v_1 depends directly pada v_2, atau
  - Terdapat jalur ketergantungan dari v_1 ke v_2.
- View v dikatakan **recursive** jika bergantung pada dirinya sendiri.

Contoh: View yang menyimpan informasi course departemen Fisika di Fall 2017

```
create view physics_fall_2017 as
    select course.course_id, sec_id, building, room_number
    from course, section
    where course.course_id = section.course_id
          and course.dept_name = 'Physics'
          and section.semester = 'Fall'
          and section.year = '2017';
```

Contoh lain: View dari physics_fall_2017 khusus ruangan di Watson

```
create view physics_fall_2017_watson as
    select course_id, room_number
    from physics_fall_2017
    where building = 'Watson';
```

7.1.4 Materialized Views

Definisi 7.2

Materialized view adalah jenis view yang disimpan secara fisik dalam database.

- Salinan fisik dibuat saat view didefinisikan.
- Jika relasi yang mendasari view diperbarui, isi materialized view dapat menjadi tidak akurat.
- Oleh karena itu, materialized view harus diperbarui secara manual atau otomatis agar tetap konsisten.

7.2 Modification of the Database

Konsep

Operasi **modifikasi database** terdiri dari:

- **Deletion:** menghapus tuple dari relasi
- **Insertion:** menambahkan tuple ke relasi
- **Update:** memperbarui nilai di tuple tertentu dalam relasi

7.2.1 Deletion

Hapus semua instruktur:

```
delete from instructor;
```

Hapus instruktur dari departemen Finance:

```
delete from instructor
where dept_name = 'Finance';
```

Hapus instruktur dari departemen yang berada di gedung Watson:

```
delete from instructor
where dept_name in (
    select dept_name
    from department
    where building = 'Watson');
```

Hapus semua instruktur dengan gaji di bawah rata-rata gaji instruktur:

```
delete from instructor
where salary < (
    select avg(salary)
    from instructor);
```

Masalah: Saat tuple dihapus, nilai rata-rata ikut berubah. **Solusi:**

1. Hitung rata-rata terlebih dahulu.
2. Identifikasi tuple yang memenuhi syarat.
3. Hapus semuanya sekaligus, tanpa menghitung ulang.

7.2.2 Insertion

Menambahkan tuple baru ke tabel `course`:

```
insert into course
values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

Atau eksplisit menyebutkan kolom:

```
insert into course (course_id, title, dept_name, credits)
values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

Menambahkan tuple baru ke `student` dengan `tot_cred = null`:

```
insert into student
values ('3003', 'Green', 'Finance', null);
```

Menambahkan mahasiswa Music yang total kredinya $>$ 144 menjadi instruktur Music bergaji 18000:

```
insert into instructor
select ID, name, dept_name, 18000
from student
where dept_name = 'Music' and total_cred > 144;
```

Catatan: Subquery dievaluasi sepenuhnya terlebih dahulu sebelum insertion dilakukan. Contoh seperti berikut akan menimbulkan masalah jika tidak demikian:

```
insert into table1
select * from table1;
```

7.2.3 Update

Naikkan gaji semua instruktur sebesar 5%:

```
update instructor
set salary = salary * 1.05;
```

Naikkan gaji 5% hanya untuk instruktur yang gajinya $<$ 70000:

```
update instructor
set salary = salary * 1.05
where salary < 70000;
```

Naikkan gaji 5% untuk yang $<$ rata-rata gaji:

```
update instructor
set salary = salary * 1.05
where salary < (
    select avg(salary)
    from instructor);
```

Using `CASE` for Conditional Updates

Jika gaji $>$ 100000, naikkan 3%; jika $\leq$ 100000, naikkan 5%:
Versi dua query:

```
update instructor
set salary = salary * 1.03
where salary > 100000;

update instructor
set salary = salary * 1.05
where salary <= 100000;
```

Versi lebih baik menggunakan *case:*

```
update instructor
set salary = case
                when salary <= 100000 then salary * 1.05
                else salary * 1.03
             end;
```

Update with Scalar Subquery

Perbarui `tot_cred` untuk setiap student berdasarkan jumlah kredit dari course yang diambil dan tidak mendapatkan nilai F:

```
update student S
set tot_cred = (
    select sum(credits)
    from takes, course
    where takes.course_id = course.course_id
          and S.ID = takes.ID
          and takes.grade <> 'F'
          and takes.grade is not null);
```

Gunakan `case` agar nilai `null` diganti 0:

```
update student S
set tot_cred = (
    select case
               when sum(credits) is not null then sum(credits)
               else 0
           end
    from takes, course
    where takes.course_id = course.course_id
          and S.ID = takes.ID
          and takes.grade <> 'F'
          and takes.grade is not null);
```

7.2.4 Update of a View

Contoh: Tambahkan tuple ke `faculty` (view instruktur tanpa salary):

```
insert into faculty
values ('30765', 'Green', 'Music');
```

Permasalahan:

- View tidak menyimpan kolom `salary`, tetapi tabel `instructor` membutuhkannya.
- Dua pendekatan:
  1. Tolak operasi `insert`
  2. Tambahkan tuple berikut ke `instructor`:

     `('30765', 'Green', 'Music', null)`

7.2.5 Some Updates Cannot be Translated Uniquely

Contoh:

```
create view instructor_info as
    select ID, name, building
    from instructor, department
    where instructor.dept_name = department.dept_name;

insert into instructor_info
values ('69987', 'White', 'Taylor');
```

Masalah:

- Departemen mana yang digunakan jika ada banyak yang berada di gedung Taylor?
- Apa yang terjadi jika tidak ada departemen di gedung Taylor?

7.3 And Some Not at All

```
create view history_instructors as
    select *
    from instructor
    where dept_name = 'History';
```

```
insert into history_instructors
values ('25566', 'Brown', 'Biology', 100000);
```

Masalah:
Tuple yang diinsert memiliki `dept_name` = 'Biology', padahal view menyaring hanya 'History' ⇒ tidak konsisten dengan definisi view.

7.3.1 View Updates in SQL

Konsep

Sebagian besar sistem SQL hanya memperbolehkan update pada **simple views**.

Syarat simple view:

- `FROM` clause hanya mencakup satu relasi dasar.
- `SELECT` hanya berisi nama atribut (tanpa ekspresi, `DISTINCT`, atau agregasi).
- Atribut yang tidak disebut dalam `SELECT` boleh diisi `NULL`.
- Tidak boleh menggunakan `GROUP BY` atau `HAVING`.

7.4 Data Definition Language (DDL)

Definisi 7.3

DDL digunakan untuk mendefinisikan struktur database.

Informasi yang bisa didefinisikan:

1. Skema untuk tiap relasi
2. Tipe nilai untuk setiap atribut
3. **Integrity constraints**
4. **Index** yang harus dipelihara
5. Informasi **keamanan dan otorisasi**
6. Struktur penyimpanan fisik relasi di disk

7.5 Create Table Construct

Konsep

Sebuah relasi SQL didefinisikan menggunakan perintah `create table`:

```
create table r (
    A1 D1,
    A2 D2,
    ...,
    An Dn,
```

```
    (integrity-constraint),
    ...
);
```

- `r` adalah nama relasi
- A_i adalah atribut dalam skema relasi
- D_i adalah domain (tipe data) dari atribut tersebut

Contoh:

```
create table instructor (
    ID         char(5),
    name       varchar(20),
    dept_name  varchar(20),
    salary     numeric(8,2)
);
```

7.6 Integrity Constraints in Create Table

Konsep

SQL menyediakan berbagai **integrity constraint** untuk menjaga konsistensi data.

Jenis constraint:

- `primary key (A1, ..., An)`
- `foreign key (Am, ..., An) references r`
- `not null`

SQL akan mencegah update apa pun yang melanggar constraint tersebut.

Contoh:

```
create table instructor (
    ID         char(5),
    name       varchar(20) not null,
    dept_name  varchar(20),
    salary     numeric(8,2),
    primary key (ID),
    foreign key (dept_name) references department
);
```

Tabel student:

```
create table student (
    ID         varchar(5),
    name       varchar(20) not null,
    dept_name  varchar(20),
    tot_cred   numeric(3,0),
    primary key (ID),
    foreign key (dept_name) references department
);
```

Tabel takes:

```
create table takes (
    ID         varchar(5),
    course_id  varchar(8),
    sec_id     varchar(8),
    semester   varchar(6),
    year       numeric(4,0),
    grade      varchar(2),
    primary key (ID, course_id, sec_id, semester, year),
    foreign key (ID) references student,
    foreign key (course_id, sec_id, semester, year) references section
);
```

Tabel course:

```
create table course (
    course_id   varchar(8),
    title       varchar(50),
    dept_name   varchar(20),
    credits     numeric(2,0),
    primary key (course_id),
    foreign key (dept_name) references department
);
```

7.7 Updates to Tables

Drop Table

```
drop table r;
```

Menghapus seluruh struktur tabel beserta isinya.

Alter Table - Add Attribute

```
alter table r add A D;
```

Menambahkan atribut `A` bertipe `D`. Semua tuple lama akan diisi `null` pada kolom baru.

Alter Table - Drop Attribute

```
alter table r drop A;
```

Menghapus atribut `A` dari relasi `r`. *Catatan:* Tidak semua DBMS mendukung fitur ini.

Alter Table - Modify Attribute

```
alter table r modify A new_datatype;
```

Digunakan untuk mengubah tipe data (domain) dari atribut `A` dalam relasi `r`.

Contoh:

```
alter table instructor modify salary numeric(10,2);
```

Mengubah tipe kolom `salary` dari `numeric(8,2)` menjadi `numeric(10,2)`.

Alter Table - Rename Table

```
alter table old_name rename to new_name;
```

Digunakan untuk mengganti nama suatu tabel dalam database.

Contoh:

```
alter table instructor rename to lecturers;
```

Tabel `instructor` sekarang akan bernama `lecturers`.

Bab 8

Entity-Relationship

8.1 Design Phases

Definisi 8.1

Proses perancangan database dilakukan dalam beberapa tahap utama:

- **Initial Phase:** Memahami dan mencirikan secara menyeluruh kebutuhan data dari pengguna database.
- **Second Phase:** Memilih *data model* yang sesuai sebagai kerangka kerja konseptual.
- **Final Phase:** Mengimplementasikan model ke dalam sistem database:
  - **Logical Design:** Menentukan skema database (struktur konseptual).
  - **Physical Design:** Menentukan tata letak fisik data di penyimpanan.

8.2 Design Alternatives

Definisi 8.2

Dalam mendesain skema database, ada dua jebakan besar yang harus dihindari:

- **Redundansi:** Desain buruk dapat menyebabkan informasi berulang. Hal ini dapat menimbulkan:
  - Pemborosan ruang penyimpanan
  - Ketidakkonsistenan data antar salinan informasi
- **Incompleteness:** Desain buruk dapat membuat bagian tertentu dari realitas bisnis menjadi sulit atau tidak mungkin dimodelkan.

8.3 Design Approaches

Definisi 8.3

Dua pendekatan utama dalam perancangan database:

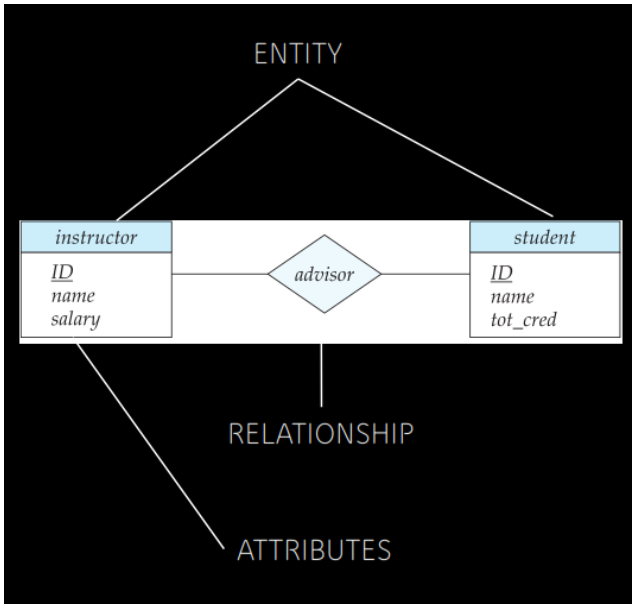
- **Entity Relationship Model (dibahas dalam materi ini):**
  - Memodelkan dunia nyata sebagai kumpulan entitas dan relasi.
  - **Entity:** Objek yang dapat dibedakan dari objek lain. Digambarkan dengan atribut.
  - **Relationship:** Asosiasi antara beberapa entitas.
  - Direpresentasikan secara grafis dalam bentuk **ER diagram**.
- **Normalization Theory (dibahas pada topik selanjutnya):**
  - Teori formal untuk mengidentifikasi desain yang buruk.
  - Menyediakan uji formal untuk memastikan kualitas skema.

8.4 ER Model - Database Modeling

Definisi 8.4

ER (Entity-Relationship) model dikembangkan untuk memudahkan perancangan database secara logis.

- Tujuan utamanya adalah untuk menentukan **enterprise schema**, yaitu representasi logis menyeluruh dari struktur database.
- ER model menggunakan tiga konsep dasar:
  - **Entity sets**
  - **Relationship sets**
  - **Attributes**
- ER model memiliki representasi visual bernama **ER diagram**, yang digunakan untuk mengekspresikan struktur logis database secara grafis.



Gambar 8.1: ER diagram

8.5 Entity Sets

Definisi 8.5

Sebuah **entity** adalah objek yang dapat dibedakan dari objek lainnya.

Contoh: seseorang tertentu, perusahaan, tanaman, peristiwa.

Entity set: sekumpulan entitas dari jenis yang sama yang memiliki properti yang sama.

Contoh: sekumpulan orang, perusahaan, liburan.

Entitas direpresentasikan dengan **atribut**, yaitu properti deskriptif yang dimiliki semua anggota entitas.

Contoh:

- `instructor` = (ID, name, salary)
- `course` = (course_id, title, credits)
- **Domain:** himpunan nilai yang diizinkan untuk atribut.
- **Primary key:** subset atribut yang secara unik mengidentifikasi entitas.

Notasi grafis:

- Entity set digambarkan sebagai **persegi panjang**.
- Atribut ditulis di dalamnya.
- Atribut yang menjadi *primary key* diberi garis bawah.

8.6 Relationship Sets

Definisi 8.6

Sebuah **relationship** adalah asosiasi antara beberapa entitas.
Relationship set: adalah himpunan relasi antar $n \geq 2$ entitas dari entity set berbeda.

Contoh:

- `advisor` adalah hubungan antara mahasiswa dan dosen pembimbing.
- $(44553, 22222) \in advisor$

Gambar 8.2: Diamond menggambarkan *relationship set*

Notasi grafis:

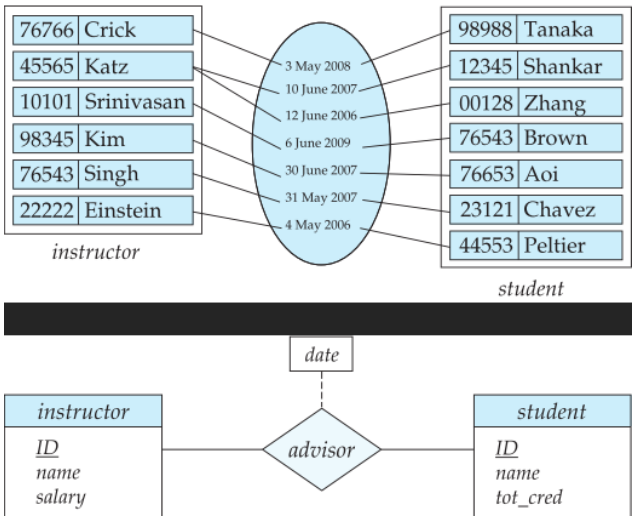
- Relationship set digambarkan sebagai **belah ketupat (diamond)**.
- Dihubungkan ke entity set dengan garis.

Definisi 8.7

Atribut juga dapat dikaitkan dengan **relationship set**.

Contoh:

- `advisor` memiliki atribut `date` untuk menunjukkan sejak kapan mahasiswa dibimbing.

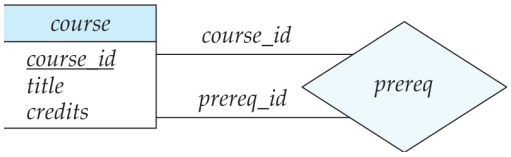


Gambar 8.3: Relationship Sets with Attributes

8.7 Roles

Definisi 8.8

Entity set yang muncul lebih dari sekali dalam satu relationship dapat memainkan **peran (role)** yang berbeda.



Gambar 8.4: Roles

Contoh:

- Dalam relationship `prereq`, entitas `course` muncul dua kali sebagai:
  - `course_id`
  - `prereq_id`
- Kedua disebut sebagai **roles**.

8.8 Degree of a Relationship Set

Binary Relationship: hubungan antara dua entity set (umumnya paling sering digunakan).
Non-Binary Relationship: hubungan antara lebih dari dua entity set.

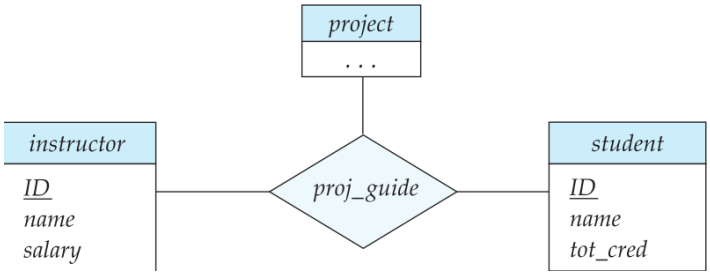
Contoh:

- Relationship `proj_guide` mengaitkan `instructor`, `student`, dan `project`.
- Ini adalah **ternary relationship**.

8.9 Non-Binary Relationship Sets

Konsep

Sebagian besar relationship set dalam database bersifat **binary** (melibatkan dua entity set). Namun, ada kalanya lebih tepat atau praktis menggunakan **non-binary relationship** (melibatkan tiga atau lebih entity set).

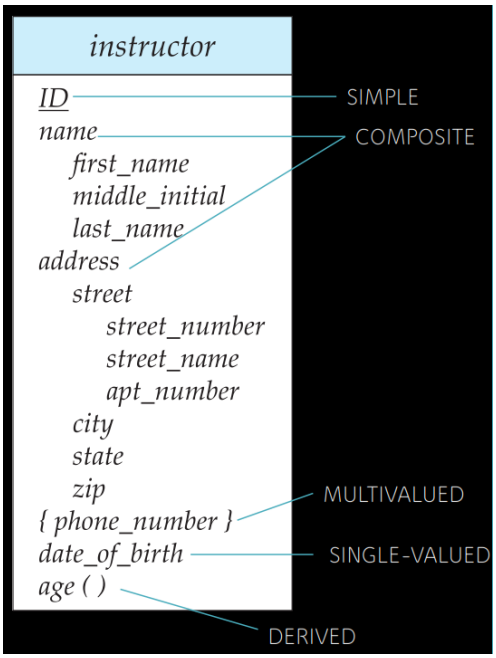


Gambar 8.5: Roles

Relationship `proj_guide` mengaitkan:

- `instructor`
- `student`
- `project`

8.10 Complex Attributes



Gambar 8.6: Complex Attributes

Jenis atribut:

- **Simple attribute:** tidak dapat dipecah lagi
- **Composite attribute:** dapat dipecah menjadi bagian lebih kecil
- **Single-valued:** satu nilai per entitas
- **Multi-valued:** bisa memiliki beberapa nilai
  - Contoh: `phone_numbers`
- **Derived attribute:** dihitung dari atribut lain
  - Contoh: `age` dari `date_of_birth`

8.11 ER Diagram with a Ternary Relationship

Konsep

ER diagram dapat merepresentasikan hubungan ternary secara eksplisit.

Notasi:

- Hubungan ternary direpresentasikan dengan **belah ketupat** yang menghubungkan tiga `entity set`.
- Bisa memiliki `atribut` yang melekat pada relationship.

Contoh: `(student, instructor, project)` dalam `proj_guide`

8.12 Mapping Cardinality Constraints

Definisi 8.9

Cardinality constraint menunjukkan berapa banyak entitas dari satu set yang dapat diasosiasikan dengan entitas dari set lain.

Jenis cardinality (untuk binary relationship):

1. One to one
2. One to many
3. Many to one
4. Many to many

Contoh penggunaan: batasan seperti "satu mahasiswa hanya bisa punya satu pembimbing", atau "satu dosen bisa membimbing banyak mahasiswa".

8.13 Cardinality Constraints in ER Diagram

Definisi 8.10

Cardinality constraints menyatakan jumlah maksimum entitas yang bisa dikaitkan dalam suatu relationship.

Notasi grafis:

- Garis **berarah (→)** = tepat satu
- Garis **tidak berarah (-)** = banyak

Contoh One-to-One:

- Satu student → satu instructor dalam relasi `advisor`
- Satu student → satu department dalam `stud_dept`

Gambar 8.7: One-to-one relationship

Contoh One-to-Many:

- Satu `instructor` → banyak `student` melalui `advisor`
- Satu `student` → satu `instructor`

Gambar 8.8: One-to-many relationship

Contoh Many-to-One:

- Banyak `instructor` → satu `student`

Gambar 8.9: Many-to-one relationship

Contoh Many-to-Many:

- Satu `instructor` ⟷ banyak `student`
- Satu `student` ⟷ banyak `instructor`

Gambar 8.10: Many-to-many relationship

8.14 Total and Partial Participation

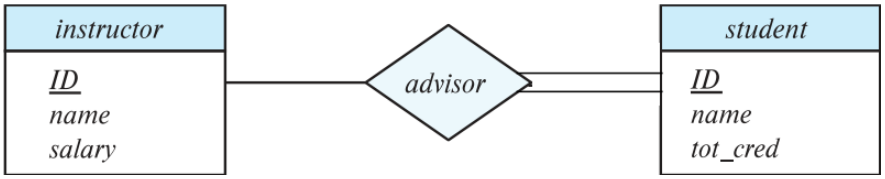
Konsep

Participation constraints menunjukkan apakah semua entitas dalam sebuah entity set terlibat dalam relationship atau tidak.

- **Total Participation:** setiap entitas berpartisipasi dalam setidaknya satu relationship (digambarkan dengan **garis ganda**).
- **Partial Participation:** beberapa entitas mungkin tidak terlibat sama sekali (digambarkan dengan **garis tunggal**).

Contoh:

- Partisipasi `student` dalam `advisor` adalah **total** → setiap student harus punya advisor
- Partisipasi `instructor` dalam `advisor` adalah **partial** → tidak semua instructor membimbing student



Gambar 8.11: Total dan Partial Participation

8.15 Notation for Expressing More Complex Constraints

Definisi 8.11

Notasi batasan kardinalitas dapat dinyatakan dalam bentuk: `l..h`, dengan:

- l = minimum cardinality
- h = maximum cardinality

Keterangan:

- Minimum 1 → total participation
- Maximum 1 → entitas hanya bisa terlibat dalam paling banyak satu relationship
- Maximum $*$ → tidak ada batasan maksimal (many)

Contoh:

- `Instructor` dapat membimbing 0..$*$ mahasiswa.
- `Student` harus punya tepat **1** advisor → partisipasi total dan kardinalitas maksimum 1.

Gambar 8.12: Complex Constraints

8.16 Cardinality Constraints on Ternary

Konsep

Untuk relasi ternary (atau lebih dari 2 entity set), maksimal hanya diperbolehkan **satu panah (arrow)** untuk menunjukkan batasan kardinalitas.

Contoh:

- Pada ternary `proj_guide`, panah dari `proj_guide` ke `instructor` berarti:
  - Setiap `student` hanya punya satu `instructor` pembimbing untuk suatu proyek.

Mengapa hanya satu panah?
Jika lebih dari satu panah digunakan (misal ke `B` dan `C` dalam `R(A, B, C)`), ada dua kemungkinan interpretasi:

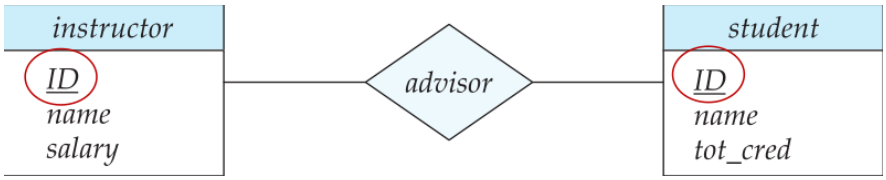
1. Setiap entitas `A` diasosiasikan dengan satu pasangan unik dari `B` dan `C`.
2. Setiap pasangan `(A, B)` diasosiasikan dengan satu `C`, dan `(A, C)` dengan satu `B`.

Untuk menghindari ambiguitas, hanya satu panah diperbolehkan.

8.17 Primary Key – Entity Sets

Konsep

Primary key digunakan untuk mengidentifikasi entitas secara unik dalam entity set.



Gambar 8.13: Primary Key pada ER Model

Karakteristik:

- Nilai atribut entitas harus cukup untuk membedakan satu entitas dengan yang lain.
- Tidak boleh ada dua entitas yang memiliki nilai atribut yang persis sama untuk semua kolom.

Key: Himpunan atribut yang secara unik mengidentifikasi anggota dari entity set.

8.18 Primary Key – Relationship Sets

Konsep

Untuk mengidentifikasi relasi antar entitas, digunakan gabungan dari primary key entitas-entitas yang terlibat.

Pilihan key tergantung pada mapping cardinality:

- **Many-to-Many:**
  - Gabungan primary key dari semua entity → minimal superkey → dijadikan primary key relationship.
- **One-to-Many atau Many-to-One:**
  - Primary key dari sisi "Many" cukup sebagai primary key.
- **One-to-One:**
  - Primary key dari salah satu entity cukup (boleh dipilih salah satu).